An Integrated Stereo Algorithm Based on Coarse-to-Fine Features and Intensity Values

by

Osama Taleb Masoud

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

June, 1994

Order Number 1359479

An integrated stereo algorithm based on coarse-to-fine features and intensity values

Masoud, Osama Taleb, M.S.

King Fahd University of Petroleum and Minerals (Saudi Arabia), 1994

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

An Integrated Stereo Algorithm Based on Coarse-to-Fine Features and Intensity Values

BY

Osama Taleb Masoud

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE
In
Computer Science

June 1994

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS
DHAHRAN 31261, SAUDI ARABIA

COLLEGE OF GRADUATE STUDIES

This thesis written by

Osama Taleb Masoud

under the direction of his thesis advisor and approved by his Thesis Committee, has been presented to and accepted by the Dean of the College of Graduate Studies, in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

Thesis Committee:

Dr. Meng Er (Chairman)

Dr. Mostafa Aref (Member)

Dr. Hussein Al – Muallim (Member)

Department Chairman

Dean, College of Graduate Studies

2.7.94
Date

To my Parents,

Brothers and Sister

Acknowledgments

All praise be to Allah for his limitless help and guidance. Peace and blessings of Allah be upon His prophet Muhammad.

Acknowledgment is due to King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, for the generous help and support for this research. I would like to express my profound gratitude and appreciation to my advisor, Dr. Meng Er, Professor of Information and Computer Science, for his guidance and patience throughout this thesis. His continuous support and encouragement can never be forgotten. I would also like to thank Dr. Mostafa Aref, Assistant Professor of Information and Computer Science, and Dr. Hussein Al-Muallim, Assistant Professor of Information and Computer Science, for their consistent support and valuable suggestions.

I also wish to thank faculty, graduate students, and the staff members of the Information and Computer Science Department for their support, especially the department chairman, Dr. Muhammed Al-Mulhem. The encouragement and good wishes of the my friends, Homam Najjari, Khaled Nassar and Tambi Baik is also worthy of acknowledgment. Finally, special thanks must be given to my family for their encouragement and moral support.

Contents

List of Tables

List of Figures

Abstract

Name: Osama Taleb Masoud

Title: An Integrated Stereo Algorithm Based on Coarse-to-Fine Features and Intensity Values

Major Field: Computer Science

Date of Degree: June, 1994

Depth perception is a major problem in computer vision. Stereo vision is one of the most useful techniques for depth perception because it gives quantitative depth measurements. Computational stereo is defined as the recovery of three-dimensional characteristics of a scene from two images taken from two points of view. In this work, a stereo algorithm is presented. Unlike most of the previous stereo algorithms, which are classified as either feature-based or intensity-based, this algorithm makes use of features as well as intensities. Features are extracted using Laplacian-of-Guassian filters of different sizes. The algorithm operates in a coarse-to-fine manner with the lowest level being the image itself to account for intensity matching. The algorithm was tested on a variety of images and gave good results in terms of accuracy as well as short running time.

Key Words: Stereo vision, Depth perception.

Master of Science Degree

King Fahd University of Petroleum and Minerals
Dhahran, Saudi Arabia

June, 1994

خلاصـة الرسالـة

اسـم الطـالـب: أسامة طالب مسعود

عنوان الدراسـة: خوارزمية متكاملة للرؤية الثنائية تعتمد على الخصائص متفاوتة الخشونة وعلى قيم شدة التوهج

التـخـصـص: علوم الحاسب الآلي

تاريخ الشهـادة: يونية، ١٩٩٤م

تعتبر عملية تحديد العمق من المشاكل الأساسية في حقل الرؤية بالكمبيوتر. الرؤية الثنائية هي من أفضل الطرق لتحديد العمق لأن بإمكانها تحديده بدقة. تعرف الرؤية الثنائية بأنها عملية استرجاع الخصائص ثلاثية الأبعاد لمنظر ما باستخدام مسقطين مأخوذين من مكانين مختلفين. يقدم هذا البحث طريقة حل (خوارزمية) للرؤية الثنائية. خلافا لمعظم الخوارزميات السابقة والتي يمكن تصنيفها بأنها إما معتمدة على شدة التوهج أو معتمدة على الخصائص العامة، فإن هذا الخوارزم يحاول الاستفادة من قيم شدة التوهج والخصائص العامة في آن واحد. تستخرج الخصائص العامة باستخدام مصفاة من نوع (Laplacian-of-Gaussian) ذات أحجام مختلفة. تعمل الخوارزمية بطريقة خشن-إلى-ناعم بحيث يكون آخر مستوى هو الصورة نفسها للقيام بعملية توفيق قيم شدة التوهج. تم اختبار الخوارزمية على العديد من الصور وكانت النتائج جيدة من حيث الدقة والسرعة.

درجة الماجستير في العلـوم

جامعة الملك فهد للبترول والمعادن
الظهران، المملكة العربية السعودية

يونية، ١٩٩٤

Chapter 1

Introduction

Artificial intelligence (AI) is concerned with the study of intelligence using computational methods. Its main goal is to build computers with capabilities comparable to those of biological organisms. Dealing with information received through sensory organs is one of the obvious biological capabilities. Of all the sensory organs, the eyes constitute the major tool facilitating communication and interaction with the external world. In fact, about 70% of information input to the human brain is through the eyes.

Computer vision aims to study the process of 'seeing'. In particular, it tries to construct rich descriptions from an image, emulating the vision process in biological systems. Computer vision, or alternatively called image understanding (IU), is a vast

field in AI covering all levels of visual processing from retinal to cognitive levels.

1.1 Applications and Focus of Computer Vision

Computer vision applications are manifold. These include: automation of industrial processes, inspection tasks, remote sensing, making computer power more accessible, military applications, and aid for the partially sighted. Although many application specific vision systems have been designed, the major focus of computer vision has been concentrated towards visual abilities rather than the domain of applications. Visual abilities refer to identifiable modules in the human visual system.

1.2 Related Fields to Computer Vision

There are many fields concerned with the processing of visual images; and thus, some of them have a close relation to computer vision. Image processing is one example of such disciplines. It mainly deals with transmission, storage, enhancement, and restoration of images. Many techniques of image processing have been adopted in the early stages of computer vision such as edge detection and region finding. Other examples include computer graphics and computer-aided design and manufacture (CAD/CAM) whose main concentration is on the display of visual surfaces and

surface representations, respectively. Computer vision finds some techniques in these disciplines particularly useful.

Finally, there is the field of pattern recognition which, although very different from computer vision, is probably the most closely related to it. While pattern recognition expects its input to fall within a set of input known in advance, deals with 2-D images, and operates directly on the image, computer vision deals with 3-D scenes which cannot be enumerated in advance and operates on a range of representations including the image itself.

That many disciplines are closely related to computer vision does not mean that computer vision is a collection of those; none of the disciplines mentioned above share computer vision its unique characteristic: it is concerned with the understanding of images.

1.3 The Human Visual System

The human brain, being the most complex structure in the known universe, has drawn the attention of scientists in a variety of disciplines. Each discipline has concentrated on a different aspect of the brain and yet, very little is fully understood so far. The visual cortex, which is the part of the brain responsible for seeing,

did get its share of attention from many fields such as: psychology, philosophy, physics, psychobiology, neurophysiology, and psychophysics. The latter two are of a particular interest to computer vision. Neurophysiology is concerned with the study of the internal components of the visual system while psychophysics concentrate on the external behavior of the visual system. For example, while neurophysiologists investigate neurons which fire when a certain stimuli is being viewed by the eyes, psychophysicists concentrate on what actually is perceived by viewing that stimuli.

Computer vision, being interested in a model similar to that of the human, has benefited from the many findings by these two disciplines. One of the most important findings is that the human visual system is modular in the sense that it consists of individual modules which operate on representations computed by other modules. These modules can be classified as either top-down (high-level) or bottom-up (low-level) processes as in figure 1.1. High-level processes have to do with semantic memory and symbolic processing; and there is very little known about their details. Knowledge about low-level processes is, on the other hand, relatively mature [Jul91].

The study of low-level processes independent of high-level processes was made possible by the fact that high-level processes do not seem to affect the operation of low-level processes. Much work has been done on identifying and studying the various low-level modules which are collectively called early vision. Early vision will be discussed in more detail in the next section. Low-level and high-level vision

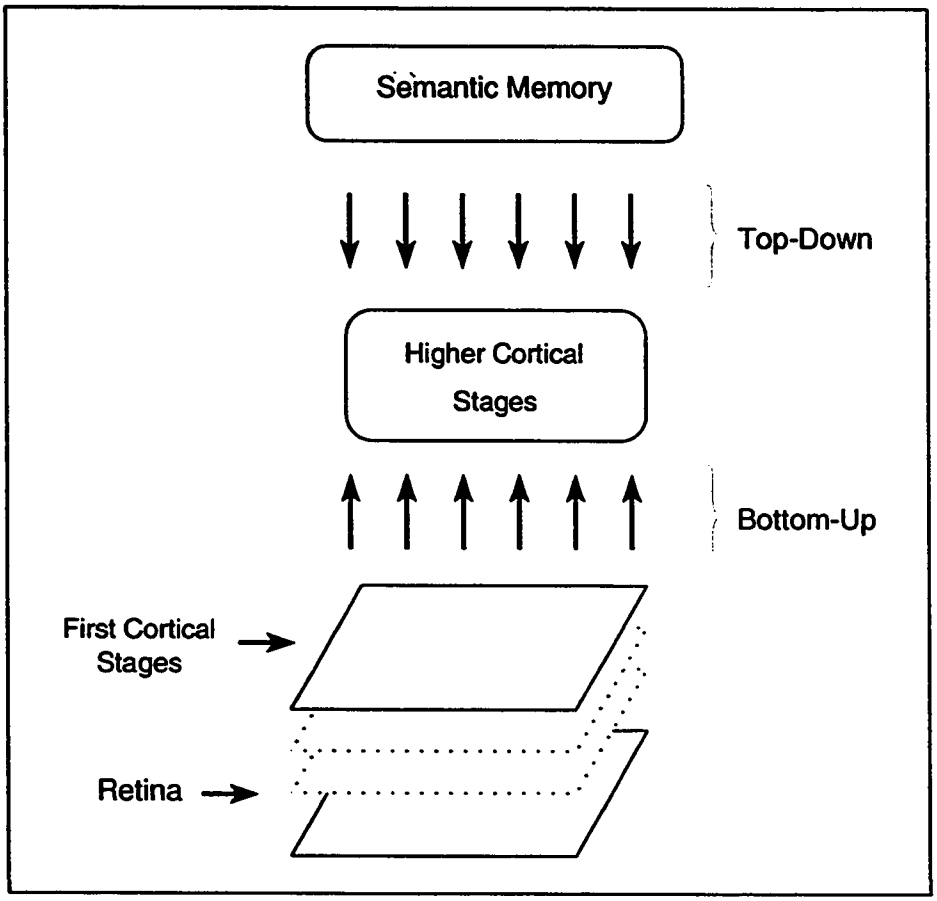


Figure 1.1: Bottom-up vs top-down processes in the visual system

are still fertile fields of research and hence the division of computer vision into a reconstruction school and a recognition school which corresponds to low-level and high-level vision, respectively.

1.4 Early Vision

Early vision refers to those modules in the early stages of vision. It is a pure bottom-up process which is not influenced by the top-down semantic processes. The input to early vision is a pair of two-dimensional projections (images) of the scene on the retinas of the two eyes. Therefore, some of the modules in early vision must operate directly on these images.

Images corresponding to natural scenes are highly redundant and misleading at the same time. There are around 23 different cues for determining depth and surface layout, many of which usually appear together in the same natural image. Somehow, the visual system uses all available cues and filters out misleading information. Therefore, it was natural to suggest that different modules, each dealing with a different type of cue, should exist.

Figure 1.2 depicts one suggested model, among many others, showing various early vision modules along with the representations on which they operate [Bra82].

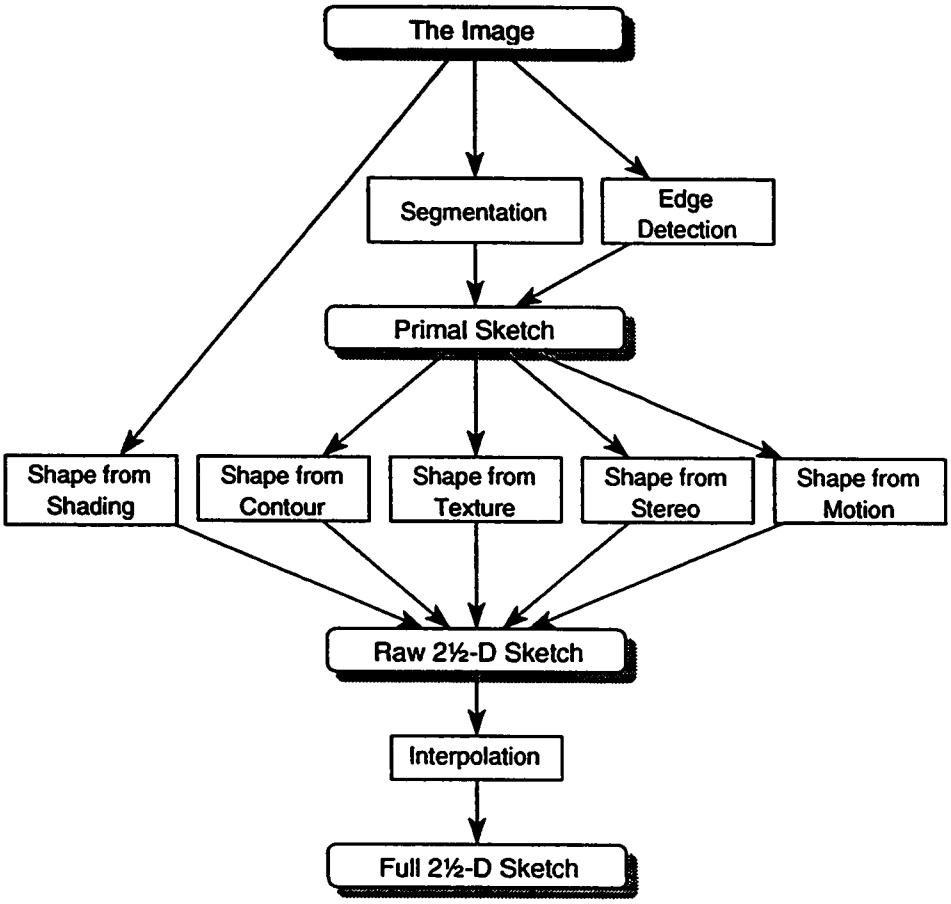


Figure 1.2: Early vision modules and representations

As can be seen, many of these modules are concerned with deriving shape from a certain cue. The four representations shown in the figure are:

- The image: the two-dimensional projection of the scene on the retina of each eye. The image can be represented as a two-dimensional array of intensity values.
- The primal sketch: similar to the image except that it only contains significant features such as edges, boundaries, etc.
- The raw $2\frac{1}{2}$-D sketch: contains surface information at the locations of the features in the primal sketch.
- The full $2\frac{1}{2}$-D sketch: contains surface information at all the image points.

The modules shown in the figure are:

- Edge detection: refers to the detection of significant features in the image such as contours, shadow boundaries, discontinuities in surface orientation, etc. These features usually exist in the image at locations where there is a drastic change of intensity. All the other insignificant information is filtered out in the process.
- Segmentation: the dual of edge detection in the scene that rather than isolating

image features, where there is a drastic change of intensity, it isolates image regions having no significant changes of intensity.

- Surface from shading: refers to the process which tries to find surface orientation using shading information (the way shading varies over the surface.)

- Shape from texture: Texture can be defined as repeating patterns of a certain shape over an area. The changes in pattern appearance (such as size) over a surface can be used to determine the orientation of that surface.

- Shape from contours: The shape of the contours in the primal sketch can be used to find the slant and tilt of surfaces in the image.

- Shape from stereo (the subject of this research): The difference between two images (a stereo pair) corresponding to the same scene but taken from two different points in space (such as the two eyes) is utilized by this module to determine the depth of features in the image.

- Shape from motion: similar to shape from stereo except that the two images are taken in two different points in time rather than space.

- Interpolation: In order to produce the full $2\frac{1}{2}$-D sketch from the raw $2\frac{1}{2}$-D sketch which contains surface information in certain locations, this information is interpolated to produce similar information for all points in the image.

1.5 Stereo Vision

Stereo vision is one of the main processes in the human visual system for determining the relative depth of objects. The process of finding such depth is called stereo fusion or stereopsis. Because the eyes are separated (by approximately 6.5cm), each eye sees a different view of the scene ahead. The difference between the images seen by the two eyes accounts for the depth of objects in the scene. This can be illustrated by the following experiment. Close your left eye and place your thumbs one behind the other in front of you. Now by opening the left eye and closing the right one, you will see that the close thumb has moved to the right relative to the far thumb. Further, your far thumb has moved to the right relative to the background but by a smaller distance. The distance by which the location of an object differs in both images (which is called the *disparity*) is what determines the depth of the object (the distance from the viewer). The disparity of the close thumb was larger then that of the far thumb, and the disparity of the latter was larger than that of the background.

1.5.1 Stereopsis and random-dot stereograms

Stereoscopic fusion was believed to occur in the human visual system after the recognition of objects in each image (a late process.) For example, in the experi-

ment mentioned above, the visual system would recognize the objects (the thumbs) and then match corresponding objects in each image. In 1960, Julesz [Jul60] demonstrated that this cannot be true. In fact, he proved that stereopsis is an early vision process. His demonstration made use of the so-called random-dot stereograms.

A random-dot stereogram is a pair of images each of which is a totally random pattern of dots and thus devoid of any monocular features (such as edges, etc.) However, when the two images are binocularly viewed (using a stereoscope, for instance), a vivid three-dimensional structure is seen. The recognition of this structure could not possibly occur before stereopsis, which shows that stereopsis must be an early process. One way to construct a simple random-dot stereogram is by filling one image (a square matrix) with dots (black or white) on a random basis. The other image is an exact copy of the first one except that a box of a certain size in the matrix is shifted to the left (or to the right) by a small distance (a column, for example.) The empty area revealed by shifting the box is filled again with random dots.

1.5.2 The correspondence problem

The problem of deciding which feature in one image matches which feature in the other image is called the correspondence problem. Of course, if the correspondence

problem is solved, all what remains is a straightforward calculation of the distance to matched features using their disparity information. The difficulty of the correspondence problem is depicted in figure 1.3. Each dot among the four dots seen by the left eye could be matched with any dot seen by the right eye. However, the human visual system settles on a single set of matches (the correct matching), namely, the matches shown in black in the figure. Finding the correct match is the solution to the correspondence problem, as it has been the aim of all stereo matching algorithms so far.

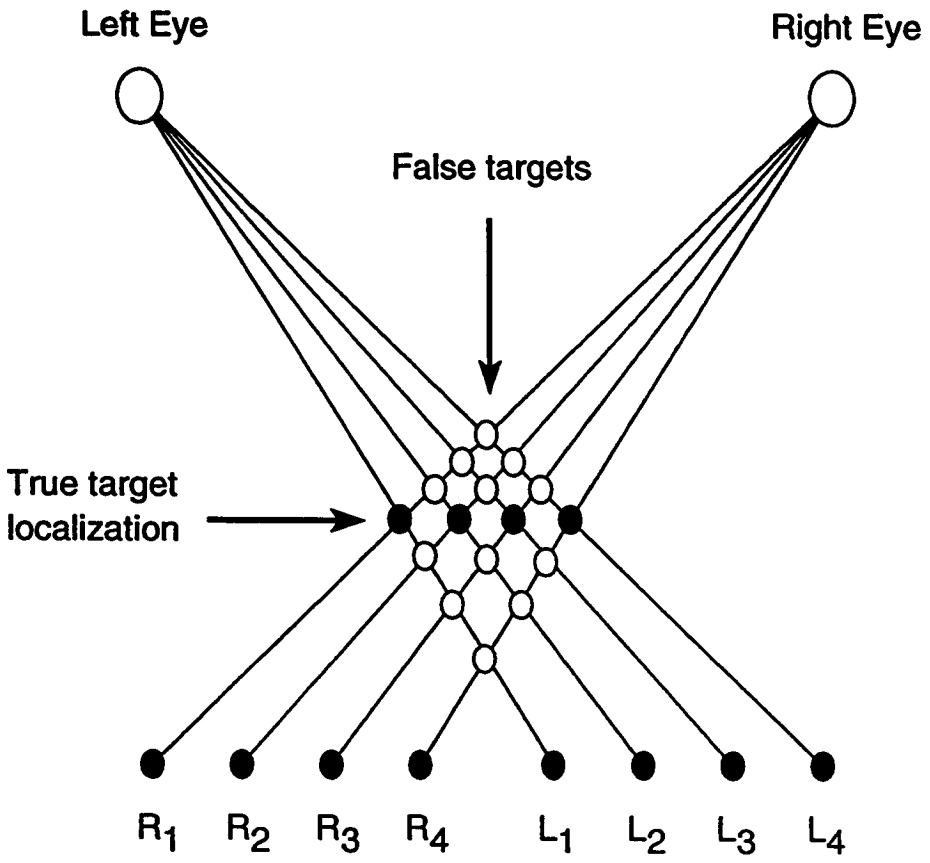


Figure 1.3: False target elimination problem. There are 16 possible matches for the 4 dots seen by each eye. Only 4 of these matches are correct.

Chapter 2

Literature Review

Ever since Julesz demonstration in 1960 [Jul60] which showed (using random-dot stereograms) that stereopsis is an early visual process, many computational theories and algorithms for stereo vision have been developed, many of which tried to maintain consistency with the processing in the human visual system. A distinction should be made between a computational theory and the underlying algorithm. A computational theory investigates symbolic representations and transformations between them whereas an algorithm refers to a set of instructions which perform the transformations; and thus, there can be more than a single algorithm for the same computational theory.

A typical stereo algorithm consists of the following stages:

1. Feature extraction

2. Matching

3. Computing the depth

Matching is the key step in the algorithm. It can be thought of as two interacting sub-problems: local matching and global matching. Local matching refers to identifying features in both images corresponding to the same point in 3-D space. The difficulty arises from the fact that the two features corresponding to the same point in 3-D space do not have to be identically similar because the two images are actually two perspective views taken from two different view points. Therefore, the requirement that features must have exact similarity should be relaxed. However, by doing this, false matches might occur by having two similar—but not identical—features which do not correspond to the same point in 3-D. Global matching attempts to solve this problem by considering sets of local matches globally and trying to resolve ambiguities.

There are several issues that any theory for stereopsis should take into account:

- Primitives to be matched.
- Assumptions about the scene.
- Determination of geometry and calibration of the stereo system.

Most algorithms differ from each other by the way they address these issues. Algorithms in literature have dealt with these issues in a variety of ways as follows.

1. **Primitives to be matched:** Images are normally represented as two-dimensional arrays of intensity values (irradiance values). Fusing the stereo pair requires that correspondence is made between primitives in the left image with primitives in the right image. The choice of the type of these primitives is crucial and has a great effect on the stereo algorithm behavior both in terms of accuracy and efficiency. This issue is directly related to local matching. Choosing intensity values as primitives is one extreme, the other extreme being the choice of scene objects as primitives.

   Intensity point extraction is straight forward; but the search space needed to find matches between primitives in both images is huge due to the fact that there can be a large number of points having the same intensity values. Furthermore, because of possible intensity distortions resulting from noise, the matching process may produce erroneous results. Algorithms which use intensity values as primitives are called intensity-based (or area-based). The other extreme of identifying objects before matching considerably reduces the search space (scenes usually consist of several objects) but the complexity involved in extracting objects is enormous. In addition, this cannot be the way stereopsis is performed in the human visual system since we know that

human stereopsis is an early process.

Between the two extremes, many types of primitives are possible. Edge detection provides one such type of primitives called zero-crossings. A zero-crossing corresponds to a significant change in intensity across the image. Zero-crossings can be used as primitives or can be combined to produce richer primitives such as contours, etc. Edge-based (or feature-based) algorithms use such primitives. Because such primitives are sparsely distributed over the image, the search space is much smaller than that used in intensity-based algorithms; and thus, feature-based algorithms tend to be faster. Most algorithms in literature are feature-based. It is believed that the human visual system uses rich primitives similar to those used by feature-based algorithms in the matching process.

2. **Assumptions about the scene:** Global matching is confronted with many ambiguous matches. To resolve these ambiguities, certain assumptions about the scene are usually necessary. For example, consider figure 1.3 in section 1.5.2. Because the primitives are all similar, local matching is of little help since any point in one image can be matched with any point on the other. Global matching has to select the best set of matches. There is no clear definition for the best set of matches except that it is the set that would be matched by the human visual system. Several assumptions have been considered in the stereo

vision literature:

- Smoothness (continuity) constraint
- Uniqueness constraint
- Opacity constraint
- Disparity-gradient constraint

The **smoothness** constraint states that the depth map of the image should be as smooth as possible, except at places where there are discontinuities in depth (such as object contours.) To impose this constraint, points in close proximity are required to have close depth values. This constraint is a direct consequence of the assumption that matter is cohesive. The **uniqueness** constraint states that every point in an image should match at most one point in the other image, which means that if the two lines of sight cross at a certain point in 3-D space, there should not be any other match on the either line of sight. The **opacity** constraint extends the uniqueness constraint by prohibiting any match lying within the hourglass-shape forbidden zone bounded by the two lines of sight. The **disparity-gradient** constraint defines a similar hourglass-shaped forbidden zone around a match. All these constraints help to cut down the number of false matches.

3. **Determination of geometry and calibration of the stereo system:** Most stereo algorithms assume that the two images are taken by two cameras

whose optical axes are parallel to each other and perpendicular to the base line (the line connecting the optical nodes). **Epipolar** constraint refers to this arrangement in which points in 3-D space which would project on a horizontal scan line in the left image would also project on a corresponding horizontal scan line in the right image. This simplifies the matching process since only points lying on a horizontal line in one image need to be searched for a match with a point in the other image. Without this assumption, rectification is needed to align the images in such a way that the assumption becomes true. Some stereo algorithms include a rectification step before performing the matching.

As we mentioned, existing stereo algorithms can be classified into area-based and feature-based. The following two sections briefly describe these two classes.

2.1 Area-based Stereo

Area-based stereo algorithms attempt to match pixels in the left image with pixels in the right image. Usually, a small area around the pixel is considered and a search is performed in the other image for best match of this area. The goodness of a match is found by a cross-correlation measure like the sum of differences of intensities. To reduce the search space, the epipolar constraint is utilized. A coarse-to-fine strategy

in which the image is processed in different resolutions from coarse to fine helps to further reduce the search space since search at fine resolution levels is guided by search at low resolution levels. Global matching is sought by imposing a smoothness constraint on the depth map.

Traditional area-based stereo algorithms require the presence of textured surfaces and suffer from their sensitivity to illumination and presence of surface discontinuities [Bra82]. Recently, Vleescauwer [Vle93] proposed an intensity-based algorithm which uses a conservative coarse-to-fine approach. Using some reliable initial matches at the coarse level, an initial depth map is estimated. The depth map is refined and unfeasible parts are removed. Interpolation is used to recover the complete disparity field. Then, this field is used to restrict possible depth maps at lower levels.

2.2 Feature-based Stereo

Feature-based algorithms use features detected using edge detection techniques. Many of these algorithms employ a coarse-to-fine strategy to reduce the search space.

Marr and Poggio [MP79] proposed a feature-based algorithm which was imple-

mented by Grimson [Gri81]. The algorithm matched zero-crossings with similar attributes (sign and orientation). The algorithm also used a coarse-to-fine strategy and assumed uniqueness and continuity. Vergence control of the eyes was done to bring portions of the image into a range where matching can take place. Matches belonging to areas where the number of matches falls below a certain threshold are not accepted. The algorithm worked well with random-dot stereograms but had some problems (such as erroneous vergence movement across sharp discontinuities) when experimenting with real images. The algorithm was refined later by Grimson [Gri85] to exploit figural continuity.

Baker and Binford [BB81] used a simple horizontal operator to detect edges. The ordering constraint was utilized in solving the correspondence problem. Matching was performed using a dynamic programming technique on a row-by-row manner. Each pair of rows are treated separately and a search is conducted to find matches between edges in these rows. This is called intra-scanline search or 2D search (the two rows can be thought of as two axes, thus 2D). The problem with this method is that it ignores neighboring scanlines during the search process. Neighboring scanlines provide useful information which can be used to guide the search process in the current scanline. For example, if in the current scanline, a point on an edge e_l in the left is matched to a point on an edge e_r in the right image, another point on e_l on the neighboring scanline will most likely have its correct matching point

on e_r. Ohta and Kanade [OK85] improved on this algorithm by extended dynamic programming to perform matching across scan lines as well. Such search is called inter-scanline search or 3D search. The errors were consequently reduced by a factor of 10. However, the time complexity of the already computationally expensive dynamic programming search increased by a factor of 10. This renders a sequential implementation of this algorithm impractical.

Medioni and Nevatia [MN85] matched straight line segments (thus utilizing the inter-scanline dependency) which are similar in the orientation and contrast. A minimal differential disparity criterion was used for global matching. The differential disparity is a measure of how the disparity of the match in question agrees with the average disparity of the matches in its neighborhood. A form of relaxation was used to collect matches gradually. The algorithm performed well in terms of accuracy and speed. However, it seems heavily dependent on the connectivity of edges. The test cases used had plenty of long connected edges.

Pollard, Mayhew and Frisby [PMF85] imposed a disparity-gradient constraint and used a relaxation technique to gradually build the disparity map. Uniqueness constraint is imposed by solving the problem from the left image to the right and vice versa; then only common matches are selected. Their algorithm, also called the PMF algorithm, gave very satisfactory results. The speed was in the range of 20 minutes for an image of size 256 by 256 pixel when run on a Sun 2 with a SKY

floating point accelerator [BD91].

Eastman and Waxman [EW87] imposed a smoothness assumption by fitting planar surface patches locally. Matches violating a disparity-gradient constraint or severely affecting the residual of the fit are eliminated. Finally, the disparity map with the minimum residual is accepted. The algorithm thus integrates the correspondence process and the process of constructing a dense disparity surface. The algorithm was tested on smooth synthetic surfaces that had little disparity variations but the errors were acceptable.

Kim and Bovik [KB88] imposed a disparity smoothness property along image contours. Initially, only a small set of specific high-information points are matched. Measured disparities are then propagated along contours to find matches for other points. This greatly reduces the computational cost.

Olsen [Ols90] proposed a multi-pass matching and reconstruction algorithm. In first stages of the algorithm, only reliable matches are considered and they are used to refine the disparity map. In later stages, the requirement that new matches are reliable is loosened as the requirement that they should agree with the disparity map is tightened.

Chen and Medioni [CM90] introduced a formalism called adaptive smoothing which preserves the location of edges across scales (coarse-to-fine), alleviating the

correspondence problem.

Yuille, Geiger and Bulthoff [YGB90] described a theoretical formulation for stereo in terms of the Markov Random Field and Bayesian approach to vision. The techniques are taken from statistical physics. An energy function is formulated and techniques to minimize it are described; thus, the stereo problem is transformed into an optimization problem.

Jordan and Bovik [JB91] introduced color as a disambiguating attribute when performing local matching. It is essentially similar to PMF algorithm except that the match extraction stage is modified to include color agreement between primitives.

Chapter 3

Theoretical Issues

Before our algorithm was designed, a theory on which the algorithm is to be based had to be developed. We tried to maintain consistency between our computational theory and the theories which were proposed to explain various aspects of the human visual system.

This chapter will discuss several theoretical aspects of our stereo algorithm. These include the geometrical model assumed by the algorithm, the feature extraction method, the motivation for using a coarse-to-fine strategy, the method used for feature global and local matching, and the method used for intensity matching.

3.1 Stereo Geometry

A stereo pair is composed of two perspective projections of a scene on two different planes. The optical nodes for these projection planes are separated by some distance. We will assume that the two images satisfy the epipolar constraint; that is, points in 3-D space which would project on a horizontal scan line in one image will also project on a corresponding horizontal scan line in the other image. This assumption is justifiable because even if the images were not aligned in this manner, there are rectification algorithms which will do the necessary transformations to bring them into alignment once the camera geometry has been identified.

The epipolar constraint enormously reduces the matching search space since a match for a point on a certain horizontal scan line in one image will be sought on a corresponding scan line in the other image rather than the whole image.

This alignment can be achieved by placing two identical cameras in such a way that their two projection planes coincide (are part of the same larger plane). In other words, the cameras optical axes are parallel to each other and perpendicular to the base line. Such arrangement is depicted in figure 3.1.

We will show that if the two projections of a point in 3-D space are identified, it is a straight forward process to compute the exact location of this point; and

therefore, the job of a stereo matcher becomes that of finding pairs of points in the left and right image planes which are projections of the same point in the 3-D space. For convenience, we will introduce the following coordinate systems. Let the focal length be F and the cameras be displaced by a distance D as in the figure. Each projection plane makes a two-dimensional coordinate system with the origin being the point at which the camera's optical axis intersects the projection plane; and the x_l-axis and x_r-axis are parallel to the base line. The y_l-axis and y_r-axis lie on the projection planes as well. The third coordinate system is three-dimensional with its origin located midway between the optical nodes. The X-axis coincides with the base line. The Z-axis is parallel to the optical axes; and the Y-axis completes the orthonormal left-handed coordinate system. By triangulation, a point given by (X, Y, Z) is projected on the left and right planes at (x_l, y_l) and (x_r, y_r), respectively, where,

$$x_l = \frac{F \cdot (X + D/2)}{Z}$$

$$x_r = \frac{F \cdot (X - D/2)}{Z}$$

$$y_l = y_r = \frac{F \cdot Y}{Z}$$

The disparity d is defined as:

$$d = x_r - x_l = \frac{-F \cdot D}{Z}$$

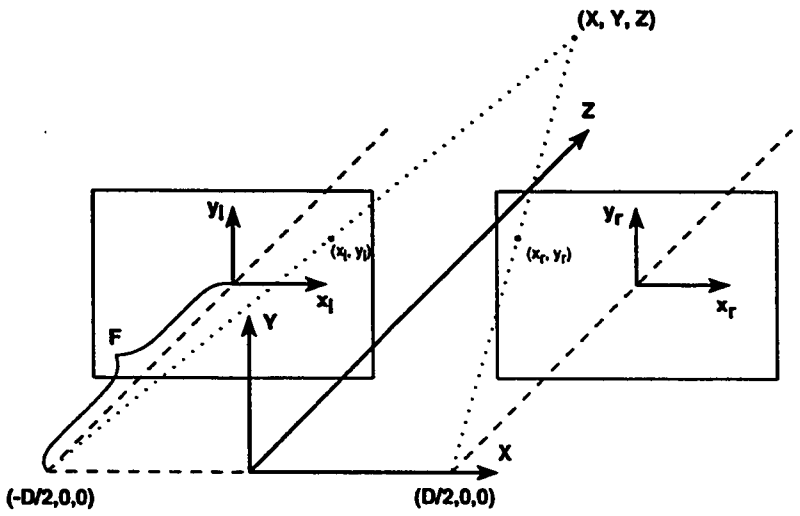


Figure 3.1: Camera model

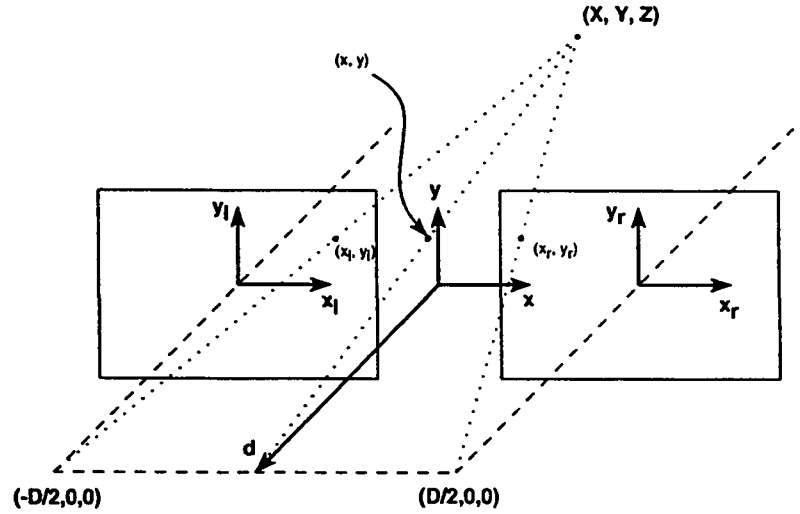


Figure 3.2: Disparity space

Figure 3.2 shows the coordinate system for the disparity space whose d-axis coincides with the Z-axis and whose x-axis is parallel to x_l and x_r axes. The xy plane is also called the cyclopean image plane. For the pair of points (x_l, y_l) on the left image plane and (x_r, y_r) on the right image plane, the corresponding point on the cyclopean image plane is (x, y) where,

$$x = \frac{x_l + x_r}{2} = \frac{F \cdot X}{Z}$$

$$y = \frac{y_l + y_r}{2} = \frac{F \cdot Y}{Z}$$

In fact, the point (x, y) on the cyclopean image plane is the projection of the scene point (X, Y, Z) on this plane if the optical node is taken to be the origin of the XYZ coordinate system. Given a point (x, y, d) in disparity space, the scene point (X, Y, Z) can be recovered using the following equations:

$$X = -x \cdot \frac{D}{d}$$

$$Y = -y \cdot \frac{D}{d}$$

$$Z = \frac{-F \cdot D}{d}$$

These two sets of equations describe a transformation from the scene space to the disparity space and vice versa. Therefore, solving the stereo problem is equivalent to finding a disparity field $d(x, y)$.

The aim of our stereo algorithm is to find this field, or at least a good approximation of it. In other words, we aim to find the disparity between pairs of points in the left and right image planes which are projections of unique points in the scene. To do this, a matching process has to be carried out in which these pairs are identified. The next section describes the nature of the points which will be considered for matching.

3.2 Feature Extraction

One of the crucial decisions in any feature-based algorithm is determining what features are to be extracted from the image because the matching will be mainly performed on these features. The two images making up the stereo pair, which are the projection of the scene on two different planes, are expected to be different. These differences are the result of the cameras being separated, noise by the cameras, and reflection properties of the surfaces in the scene. If we attempt to match everything in the left image with everything in the right image, the matching process will be misguided by differences resulting from noise and reflection properties of the surfaces. Unless these differences are taken care of somehow, their inclusion in the matching process should be avoided. Unfortunately, it is impractical to take care of reflection properties of the surfaces which require a pre-knowledge of the nature of

surfaces of the scene. It is also impractical to mask off the camera noise.

We are concerned with the differences which are the result of camera separation because these are the geometric differences. Object boundaries and texture on surfaces are examples of image features which will differ geometrically in the two images. Thus, it would be desired to extract these as matchable features and leave out the rest of the features which cannot be matched.

Most of these features correspond to locations where there is a radical change in image intensity values. Detecting such locations in each image will give us matchable features. There are various methods for detecting radical changes in intensity values in an image. Most of them deal with the image as a two dimensional surface, $z = E(x, y)$, where z is the intensity value at (x, y). Then, they find the second derivative of this surface in the x and y directions. Figure 3.3 illustrates this process in the one-dimensional case. It can be seen that the sharp change in intensities corresponds to an extremum in the first derivative and a zero-crossing in the second derivative. Thus, the zero-crossings, being easier to find than extrema, can be used as features.

In the two-dimensional case, a non-directional second-order differential operator, namely the Laplacian (∇^2), is often used. This operator is given by

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$

For practical reasons, it is better to apply a smoothness operator to the image before applying the Laplacian. This will reduce sensitivity to noise and quantization of irradiance. A good smoothness operator is the Guassian function:

$$G_\sigma(x,y) = \frac{1}{\sigma^2} e^{-(x^2+y^2)/2\sigma^2}$$

where σ is the width of the operator.

Figure 3.4 shows a three-dimensional plot of the Guassian operator which is a rotationally symmetric function. Once the image is convoluted with the Guassian, the Laplacian can be applied to the convoluted image resulting in the filtered image:

$$f(x,y,\sigma) = \nabla^2[G_\sigma(x,y) \circ E(x,y)]$$

where $\circ$ is the convolution operator and $E(x,y)$ is the image function.

Using the derivative rule for convolutions, the above expression can be rewritten as

$$f(x,y,\sigma) = \nabla^2 G_\sigma \circ E(x,y)$$

The convolution operator $\nabla^2 G_\sigma$ is given by

$$\nabla^2 G_\sigma(x,y) = \frac{x^2+y^2-2\sigma^2}{\sigma^6} e^{\frac{-(x^2+y^2)}{2\sigma^2}}$$

which is again a rotationally symmetric function with one free parameter σ. A three-dimensional plot and a cross-section of this operator which is called Laplacian

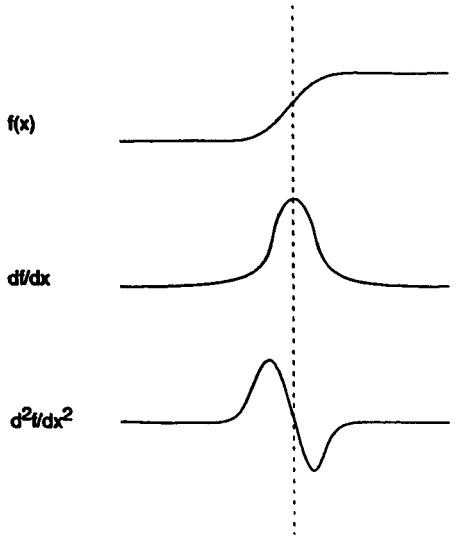


Figure 3.3: A rapid change in $f(x)$ results in an extremum in the first derivative and in a zero-crossing in the second derivative.

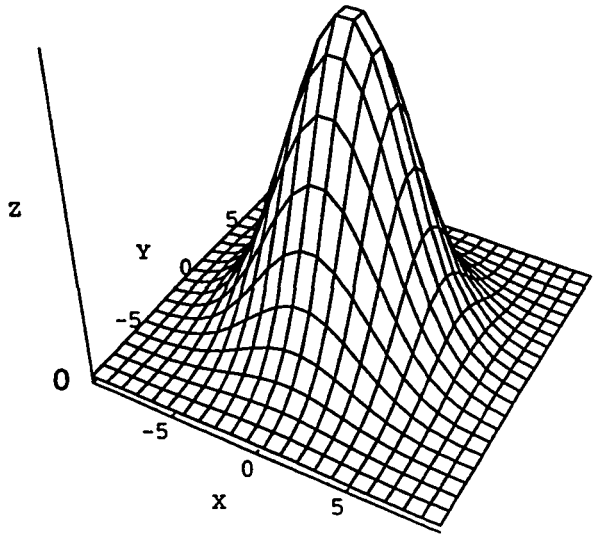


Figure 3.4: The Guassian operator.

of Guassian (LoG or Mexican hat) is shown in figure 3.5. The width of its central negative region is given by

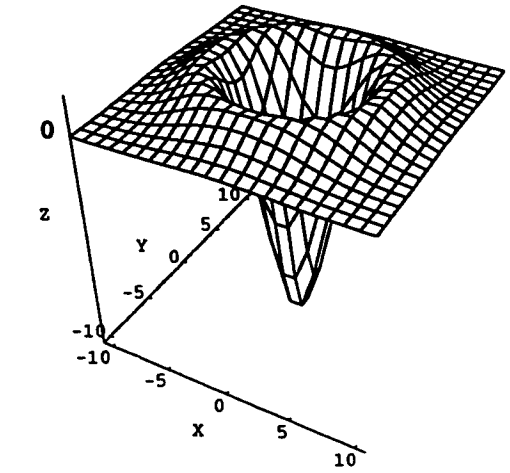
$$\omega = 2\sqrt{2}\sigma$$

Thus, the operator width can be controlled by the value of σ.

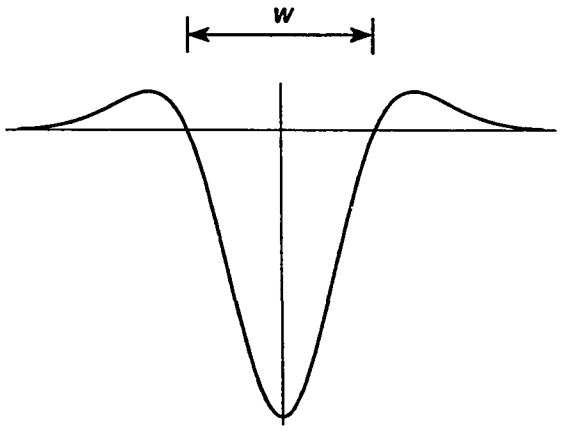
The zero-crossings of the filtered image are what we are after. To find these, we seek points (x_0, y_0) where $f(x_0, y_0, \sigma) = 0$. The symbolic representation of the image which contains only the zero-crossings is referred to as the raw primal sketch. Since the image function is discrete, f will also be discrete (i.e., has values at (x, y) where x and y are whole numbers.) The zero-crossings are more likely to occur at points in between. Therefore, it would be necessary to interpolate the values of f to get a full description of the surface.

Choosing σ a large number makes the Gaussian smoothness operator wide which will in turn blur the image with a high degree. This results in the disappearance of all the fine details in the image. The detected edges will be only those which define the broad features of the image. Conversely, a small σ will cause the operator to detect the detailed features. Thus, the degree of detail can be easily controlled by the choice of σ. 3.6 shows a sample of an image and several of its primal sketches obtained using filters of different sizes.

It is interesting to note that the use of this operator conforms with neurophysi-



(a)



(b)

Figure 3.5: The Laplacian-of-Guasian operator.

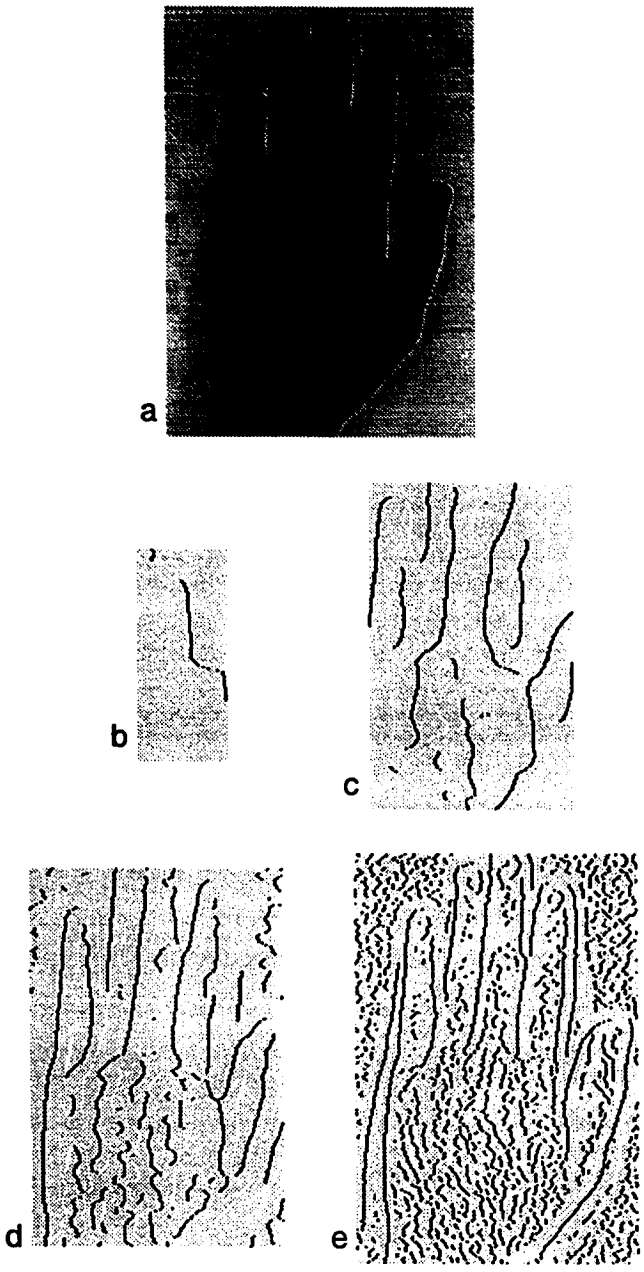


Figure 3.6: A natural image and four primal sketches obtained by convoluting the image with Laplacian-of-Guassian operators of different sizes.

ological findings. In fact, a Mexican-hat-shaped receptive field profiles were discovered in retinal ganglion cells of the human eye [KN77]. This coincidence made the use of this operator plausible especially in those algorithms which aim to mimic the operation of the human visual system.

3.3 Coarse-to-fine Strategy

In the previous section, the process of edge detection was explained. Feature-based matching is performed on the primal sketch which contains these edges. Fine primal sketches contain a very detailed set of features. If we attempt to start the matching process at a pair of fine primal sketches, we will end up with many possible matches in the right primal sketch for every match in the left primal sketch. This is costly and it is likely that many false matches can be accepted. Coarse primal sketches, on the other hand, contain gross features in the image. Because these are few, it becomes easier to match them reliably. We propose to use a coarse-to-fine approach during the matching process. This is done by having several primal sketches of the image pair, each obtained by convoluting the image pair with a filter of different size. The filter sizes will be ranging from a small value to a large value. Thus, the primal sketches will range in the degree of detail they contain. Coarse primal sketches will contain a small number of zero-crossings compared to fine primal sketches. The

matching process will start at the coarsest level. The results of this level will be used to aid the matching process at the next level and so on. This means that if a match is recorded between a zero-crossing l_0 in the left image and another r_0 in the right image, a match in the next finer primal sketch for a zero-crossing l_1 which is close in position to l_0 will be sought in an area close to r_0. This results in a reduction in the search space in the level below.

There are two reasons in favour of a coarse-to-fine strategy:

- Because the coarse primal sketches contain few zero-crossings, there will be much less false targets compared to those that would exist in fine primal sketches. Therefore, the matching process will be more accurate in avoiding false targets. This avoids being trapped in local minima. Furthermore, because of the disparity propagation process, the search area for a match at lower levels will be limited which in turn reduces the number of false targets at lower resolutions as well.

- The matching process is speeded up by restricting the number of possible matches.

We have mentioned that the human visual system contains special cells whose operation resembles that of convolution with the Mexican-hat-shaped filter. Furthermore, there is psychophysical evidence that there exist separate spatial-frequency

tuned channels in the human visual system [WB79]. Each of these acts as a filter of a different width. The studies gave evidence for four different sized channels; but it has been proposed later that an additional smaller channel may also be present. We propose to use four filters with ω values of 4, 9, 17 and 35 which agree with the channel width values found in the human visual system.

A drawback of this strategy is that if a wrong match is accepted at the coarse level, it could misguide the matching process at lower levels because the search area may no longer contain the correct match. This is an inherent problem but we attempt to minimize its occurrence by considering a set of matches in the coarse level rather than a single match. A neighborhood of a certain size is defined around the position of l_1 and all the matches of this neighborhood in the coarse level participate in the determination of the area in which l_1's partner is to be sought. This propagation process of disparity will be explained in more detail in the next chapter.

3.4 Feature Matching

We now turn to explain how features (zero-crossings) can be reliably matched. We saw earlier that matching consists of two interacting subproblems: local matching, and global matching. Local matching deals with finding pairs of features which look similar enough to be considered for global matching. Global matching attempts to

select the correct subset of these pairs by trying to resolve ambiguities on a global basis.

In this section we address the subproblem of local matching of zero-crossings. For a zero-crossing in the left image z_l, a search will be made in the right image for a zero-crossing z_r which is likely to be the correct match for z_l. The match is considered correct if z_l and z_r are the projections of the same edge in the scene. To guide this search, zero-crossings need to be given some attributes. Then, if the attributes of z_l and z_r are close enough, they can be recorded as a candidate match.

We propose to use two types of attributes for zero-crossings: the sign and local orientation. The sign of a zero-crossing refers to the direction by which the convoluted image crossed the zero (i.e. positive to negative or negative to positive) when moving along the positive x direction. This sign tells whether the edge was because of a bright area to its left and a dark one to its right or vise versa; and thus, can be considered a powerful distinguishing attribute. Zero-crossings with opposite signs cannot be matched. The local orientation refers to the direction of the edge to which the zero-crossing in question belongs. This direction is calculated locally around the zero-crossing and is recorded in degrees. Local orientation is also a powerful attribute because the edges are not expected to change much in direction. Therefore, only those zero-crossings whose local orientation is close should be accepted as candidate matches.

Our algorithm performs local matching in several passes collecting matches in a gradual manner. In the first passes, strict requirements on orientation agreement between zero-crossings are enforced. This is to guarantee that only reliable matches are accepted at this stage. These requirements are relaxed after each pass to allow for more matches to be accepted. However, the requirement that these matches agree with the previously accepted matches is tightened.

Enforcing these requirements largely reduces the number of false targets. However, the number of candidate matches left for a single zero-crossing in the left image can still be large; and that is when global matching comes in handy. The details of local matching and global matching will be discussed in more detail when the implementation details are presented in the next chapter.

3.5 Intensity Matching

Intensity (or pixel) matching is based on the assumption that the pair of images differ photometrically (in contrary to feature matching which assumes that the images differ geometrically). This assumption is far less realistic than the assumption made by feature matching; and thus, the results of intensity matching will be expected to be less accurate than those of feature matching. However, in certain cases, intensity matching performs better than feature matching. In some of these cases, where there

is a slow gradual change of intensity across a surface, feature matching becomes handicapped. That is because very few edges, if any, can be detected. We propose to use intensity matching is such cases where feature matching is ineffective.

Intensity matching is done by comparing a neighborhood of pixels in the left image with neighborhoods in the right image and selecting the one which will optimize a correlation relation. In the current implementation, the correlation relation is taken to be the sum of differences between corresponding pixels in the two neighborhoods.

In our algorithm, intensity matching is done as a final stage after the process of feature matching for the finest primal sketch is performed. Areas having few zero-crossings are marked as candidates for intensity matching. Then, pixels in these areas are matched in a way that will agree with the matches reported by feature matching. The way intensity matching is incorporated in our feature matching algorithm will be discussed in the implementation details in the next chapter.

3.6 Disparity Gradient

In this section, we introduce the concept of disparity gradient limit which will be used in the global matching part of our algorithm. This concept was first exploited

by Polard et al. [PMF85] in their algorithm known as PMF. It is based on a psychophysical finding by Burt and Julesz [BJ80].

The phenomenon of binocular rivalry (or diplopia) refers to the situation in which two (or more) objects cannot be binocularly fused by the human visual system at the same time. If one of the objects is fused, the second will appear double and vice versa. This can be easily noticed if one tries to look at two objects separated by a large distance but being behind one another with respect to his/her eyes. If he/she focuses on the near object, the far one will appear double and the other way around. This phenomenon was explained by setting a limit on the so-called disparity gradient. When the two objects are situated in a position in such a way that their disparity gradient exceeds this limit, one of them will appear diplopic (double). More formally, if the two objects are located at (x_1, y_1, d_1) and (x_2, y_2, d_2) in disparity space, the disparity gradient is defined as

$$DG = \frac{|d_1 - d_2|}{s}$$

where s is the cyclopean distance between the two points:

$$s = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

The study has shown that the disparity gradient limit for the human visual system is approximately 1 [BJ80].

It has been suggested that the human visual system introduces this limit to assist

in solving the correspondence problem. In most naturally occurring scene surfaces, the disparity gradients between the correct matches are usually less than 1. The concept of disparity gradient limit can be incorporated in our matching algorithm by giving those matches which satisfy a certain disparity gradient limit with other matches in their neighborhood a better chance to be accepted than those which do not. In the first stages of the algorithm, a strict disparity gradient limit is enforced. Then, this limit is increased gradually. The logic for doing this is similar to that of doing the relaxation process in local matching. The details of how this can be achieved will be explained in the next chapter.

3.7 Uniqueness and Ordering Constraints

It is very unlikely that two different points in the scene might project into a single point in one image of the stereo pair and into two points in the other image. Hence, a feature in one primal sketch can be restricted to be matched with at most one feature in the other image. This restriction, called the uniqueness constraint, is imposed by our algorithm by marking the features participating in an accepted match as selected.

The ordering constraint has to do with another type of unlikely events. Assume that two scene points p and q project into p' and q' in the left image plane and into p''

and q'' in the right image plane. Assume further that all projection points fall on the same horizontal scan line and p' is located to the left of q'. The ordering constraint becomes violated if p'' is located to the right of q''. This is actually impossible if the two scene points are part of the same opaque surface. Order reversal can happen in scenes which contain very small surfaces such as thin wires. The ordering constraint is based the assumption that such cases are rare and thus can be overlooked. Our algorithm enforces the ordering constraint by rejecting any match composed of two zero-crossings which are not in order with the previously accepted matches.

Eventhough by enforcing these two constraints the number of possible scenes is restricted, the gain of avoiding a large number of false targets renders this restriction justifiable.

Chapter 4

An Integrated Stereo Algorithm Based on Coarse-to-Fine Features and Intensity Values

The previous sections were concerned with theoretical considerations which describe a general framework for the algorithm. We now turn to the discussion of the algorithm details which are based on the concepts introduced in the previous sections. Of course, there can be many implementations which fit into that general framework. This algorithm was designed with efficiency being a main factor but not on the cost of robustness.

Our stereo matcher is mainly a feature-based algorithm. In the previous chapters, the merits of feature-based algorithms over area-based algorithms were discussed. However, there are cases where feature matching can never succeed in the reconstruction process. An example would be a stereo pair of a featureless sphere. The features extracted from this image will be merely two circles forming the bounding edges of the two spheres in each image. Matching these two, however accurate, will result in the reconstruction of a ring. The sphere itself can be recovered only if the intensity values are matched as well. Therefore, we cannot overlook the importance of area-based matching. In fact, matching of features as well as intensity values is necessary for the matching process to be complete. Our stereo matcher combines both in an attempt to accomplish a complete recovery of the original scene.

Figure 4.1 gives a general paradigm for the control structure of our algorithm. Having done edge detection with multiple filters, the four resulting primal sketches will be operated on during the rest of the algorithm except when intensity matching is performed. The algorithm proceeds from coarse to fine and thus operates sequentially on the primal sketches with the coarsest being the first to operate on. An example of a stereo pair and its four primal sketches on which matching will be performed is shown in figure 4.2. The operation on each primal sketch is done through four similar passes. In each of these passes, the primal sketch is traversed in a row-wise manner, collecting matches as appropriate. The first passes tend to collect

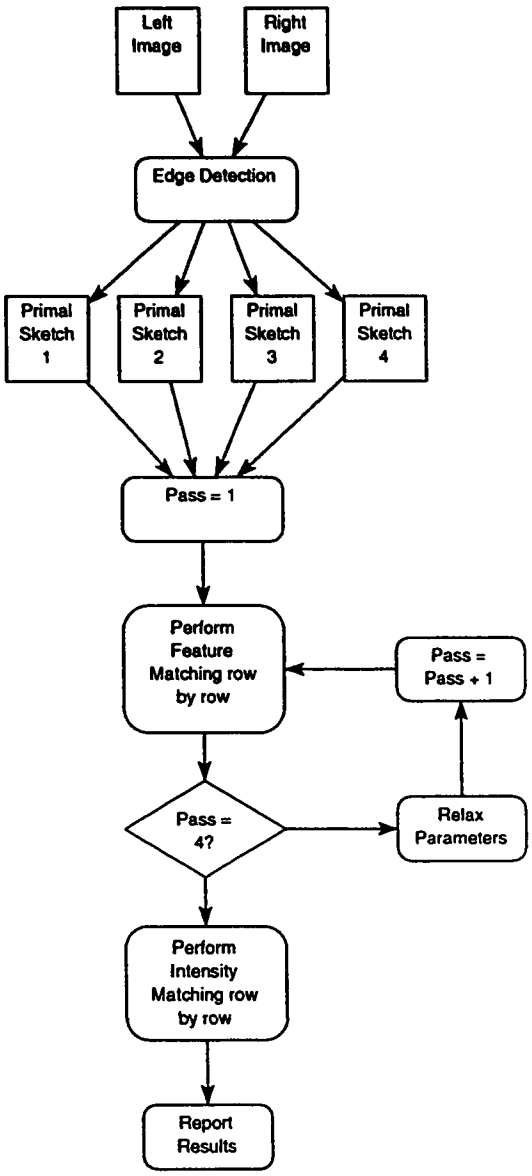


Figure 4.1: General control structure.

only reliable matches; while later passes relax some requirements so to accept more matches. However, the matches that have already been accepted by early passes are used to guide the matching process in later passes to guarantee that the new matches will also be reliable. In the last step of the algorithm, intensity matching is performed in areas where features are scarce. Guided by the set of matches that has been collected, intensity matching collects more matches to recover the structure of these previously obscure areas. The following sections will describe each part of the algorithm in more detail.

4.1 Edge Detection

In section 3.2, we gave a brief introduction to the theory of edge detection. We also mentioned that the Laplacian-of-Guassian operator will be used to perform edge detection in our algorithm. Here we explain how the primal sketch is obtained from the convoluted image; and the methods we have used to calculate the sign and local orientation attributes of edges.

The convoluted image $f(x, y)$ can be described as a surface which intersects the xy plane in various contours. These contours define the zero-crossings we aim to find. In the current implementation, this is done as follows. Each row (y is fixed) of the convoluted image is scanned from left to right (x varies from 0 to maximum). In

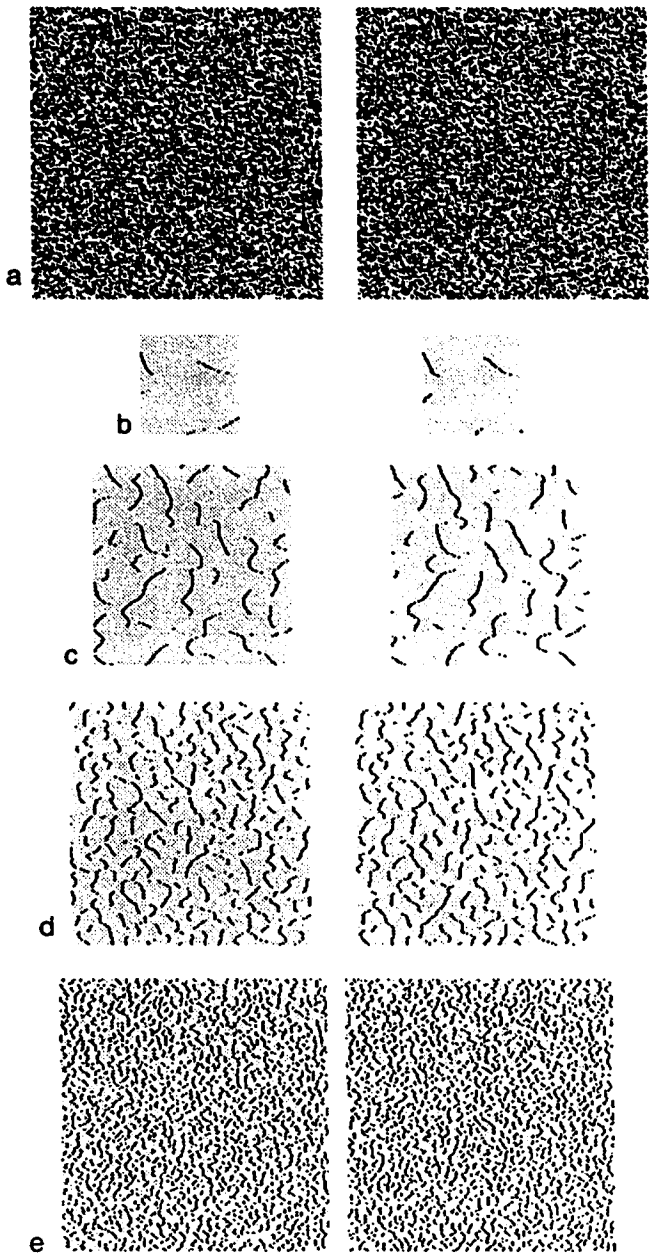


Figure 4.2: A random dot stereo pair and four primal sketches obtained from each member using filters of sizes 36, 17, 9 and 4.

this search, we find any two consecutive points (x_1, y) and (x_2, y) where $f(x_1, y)$ and $f(x_2, y)$ have opposite signs or three consecutive points (x_1, y), (x_2, y) and (x_3, y) where $f(x_2, y)$ is zero and $f(x_1, y)$ and $f(x_3, y)$ have opposite signs. In the latter case, we conclude that point (x_2, y) is a zero-crossing. In the former case, we find x_c value at which the line connecting $(x_1, y, f(x_1, y))$ and $(x_2, y, f(x_2, y))$ intersects the xy plane. Thus,

$$x_c = x_1 + \frac{f(x_1, y)}{f(x_1, y) - f(x_2, y)}$$

Then, x_c is considered a zero-crossing. By doing this, zero-crossings are detected with subpixel precision, i.e. x_c need not be an integer and thus the zero-crossing can be located between pixels. This gives more accurate disparity values (after matching is done.)

After performing this operation over all rows, we end up with a list of the detected zero-crossings in the whole image. This list is the symbolic representation known as the primal sketch. Four primal sketches for each of image of the stereo pair are constructed. The filter widths used are 4, 9, 17 and 35.

As we have mentioned previously, features (zero-crossings) have to be attributed so that local matching can be performed. The first attribute is the sign. Each zero-crossing z is attributed a sign, denoted as $\psi(z)$, which is the surface gradient sign across that zero-crossing in the x-direction. The other attribute is the local orientation, denoted as $\phi(z)$, which is calculated in degrees which range from -90

to 90. zero-crossings falling on a perfectly vertical edge will have a local orientation of 0. The angle increases as the edge is rotates in a clock-wise manner and decreases if counterclockwise. Zero-crossings on horizontal edges will have an orientation of 90. To compute this, the position of the zero-crossing in question, the position of the nearest zero-crossing in the row above, and the position of the nearest zero-crossing in the row below are interpolated using a polynomial of the second degree. The slope of the polynomial at the middle zero-crossing is computed and the arc-tangent of that is used as the orientation. If the zero-crossings in the row above (or below) is missing, the slope will be that of the line connecting the remaining two positions. If both zero-crossings (above and below) are missing, an angle of 90 is recorded. Let the zero-crossing in question be located at (x, y). Further, let the zero-crossings on the row above and below be located at $(x_a, y-1)$ and $(x_b, y+1)$, respectively. The slope of a second-degree polynomial that passes through these three points is calculated at (x, y) as

$$\frac{x_a - x_b}{2}.$$

The orientation in degrees is

$$arctan(\frac{x_a - x_b}{2}) \cdot \frac{180}{\pi}.$$

This is repeated for every zero-crossing in the primal sketch. With this, the process is complete and every zero-crossing have enough information for the matcher to start its operation.

4.2 Multiple Passes

The matching process starts at the coarsest level. For each resolution level l, primal sketch pairs which contain attributed zero-crossings are passed to the matcher. In the first pass of the matching process, strict requirements are enforced. These are:

- Zero-crossings can be matched only if the difference in their local orientations ($\Delta\theta$) is small.
- A small disparity gradient limit (DGL) is used for global matching.

In the current implementation, $\Delta\theta$ is set to 10° and DGL is set to 0.6. After each pass, $\Delta\theta$ is constantly incremented by 10° and DGL is constantly incremented by 0.2. These parameters were empirically decided after several tests were performed. The justification for using these parameters is that we want only reliable matches to be accepted in the first passes. Therefore, we start with a small $\Delta\theta$ to guarantee close agreement between matched zero-crossings and a small DGL to guarantee that only matches corresponding to points on smooth surfaces are accepted. This decreases the chance of a match being erroneously accepted. The parameters are then slowly relaxed until some maxima are reached. We use a maximum $\Delta\theta$ of 40 which is large enough to acount for almost all possible variations in orientation differences. Also, a maximum DGL of 1.2 is reached in the fourth pass which is large enough to allow

for jaggered surfaces. It has been noticed that choosing different values than these had a slight effect on the algorithm performance as long as the extremes are avoided.

The pair of primal sketches are operated on in a row-wise manner. Corresponding rows of zero-crossings are passed to the row matching routine.

4.3 Matching Rows of Zero-crossings

Each row of the two rows contains a number of zero-crossings. Some of the zero-crossings in one row have mates in the other row and thus should be matched. The rest of the zero-crossings, like those belonging to occluded surfaces in the other image, should not be matched. Figure 4.3 gives an overview of the row matching process. A bipartite graph is constructed whose left and right part vertices correspond to left and right row zero-crossings. We will use the terms zero-crossing and vertex interchangeably. Each vertex p is augmented a pair (p_x, p_y) corresponding to the zero-crossing position in the primal sketch. The disparity D between a pair of vertices p and p' is given by:

$$D(p, p') = p'_x - p_x$$

An edge connects vertex p and p' if and only if the following requirements are met:

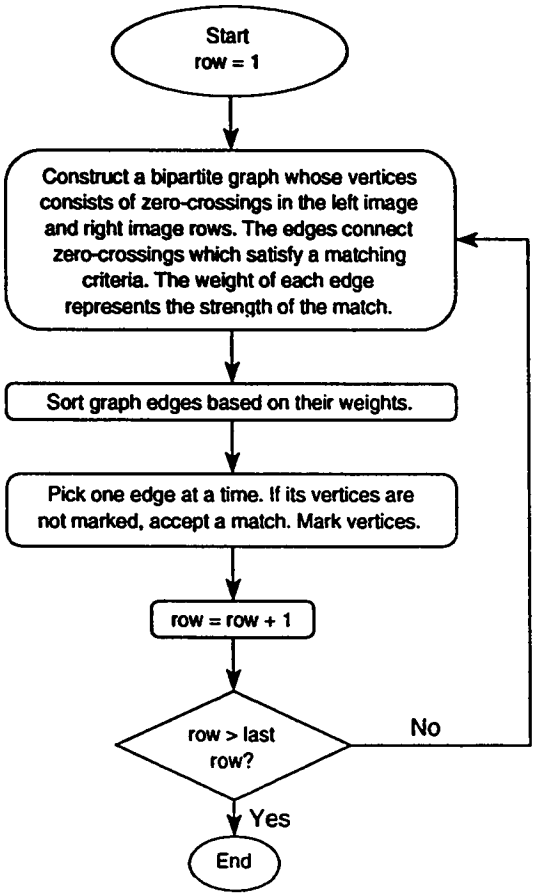


Figure 4.3: Row matching control structure.

1. The signs of p and p' are identical:

$$\psi(p) = \psi(p')$$

where $\psi(p)$ is the sign of p.

2. The difference in orientation between p and p' is less than or equal to $\Delta\theta$ + *THETA_SLACK*:

$$|\phi(p) - \phi(p')| \leq \Delta\theta + THETA\_SLACK$$

The slack of *THETA_SLACK* degrees is used as an attempt to avoid false targets. It means that zero-crossings with up to *THETA_SLACK* degrees of orientation difference above the maximum allowed difference will still be considered and will compete with the other matches but will not be selected as we will see below. In the current implementation, *THETA_SLACK* is taken to be 10.

3. The disparity between p and p' is within an acceptable range ($Disp\_range_l$) of a projected disparity. This projected disparity is computed from the disparity maps found for primal sketches of higher levels. The projected disparity for p is denoted as $PD(p)$. So this requirement becomes:

$$PD(p) - Disp\_range_l \geq D(p,p') \geq PD(p) + Disp\_range_l$$

$Disp\_range_l$ is dependent on the current resolution level l. In the current implementation, it is given values equal to the edge detection filter width used

for the current level. The way $PD(p)$ is calculated is shown below.

An edge between vertices p and p' indicates that these two vertices are a candidate match. These candidate matches are often conflicting in the sense that they cannot be satisfied simultaneously. In other words, the graph is general (neither one-to-one nor onto.) To select reliable matches among others, we need to assign a weight to each edge indicating the goodness of the match. This is done through the concept of local support based on the concept of disparity gradient limit which will be explained in section 4.3.2. Assuming that this has been done, the edge with the highest weight can be selected right away as an acceptable match. The edge with the next highest weight can be selected next; and so on. To guarantee that a vertex is not matched with more than one vertex in the other part of the graph (uniqueness constraint), vertices connected by an edge should be marked once the edge is selected. Edges connecting at least one marked vertex should be skipped. Edges whose vertices orientation difference fall outside the maximum allowed difference $\Delta\theta$ will also be skipped. But their vertices will still be marked. A final check is done on the accepted match to guarantee that it satisfies the ordering constraint. This check is conducted by comparing the current match with the two already accepted adjacent matches (if any). If the ordering constraint is violated, the match is rejected.

This is easily implemented by sorting down the edges based on their weights and then sequentially selecting them and marking the vertices as appropriate.

4.3.1 Projected disparity calculation

The motivation for using a coarse-to-fine strategy was to be able to guide the matching process at a certain level using the results of the matching process at the level above. We have mentioned that a projected disparity value $PD(p)$ is calculated for a vertex p which will determine the possible vertices to which p can be connected. $PD(p)$ is actually an approximation of the disparity value at (p_x, p_y) in the resolution level $l-1$. If l is the coarsest level, $PD(p)$ is taken to be zero. $PD(p)$ is calculated as the average of weighted disparity values in a defined circular neighborhood M around (p_x, p_y). Disparity values d_i for zero-crossings in the $l-1$ level which fall within M are given weights which are conversely proportional to the zero-crossing distance from (p_x, p_y). Let this distance be s_i. Then, the projected disparity will be

$$PD(p) = \frac{\sum_{i \in M} \frac{d_i}{s_i}}{\sum_{i \in M} \frac{1}{s_i}}.$$

The weights give more influence to disparities of close zero-crossings.

4.3.2 Calculating local support

The concept of local support was introduced to aid in resolving ambiguities. Local support of a match reflects how much this match, if accepted, will satisfy the disparity gradient limit with its neighboring matches. In the current implementation, for the match under consideration $M_{pp'}$ (consisting of a zero-crossing p in the left primal sketch and another p' in the right primal sketch), a circular neighborhood N around p is defined. We define a function DG_TEST as:

$$DG\_TEST(p,q) = \begin{cases} 1 & \textit{iff } DG(M_{pp'}, M_{qq'}) \leq DGL \\ 0 & \textit{iff } DG(M_{pp'}, M_{qq'}) > DGL \end{cases}$$

where $DG(M_{pp'}, M_{qq'})$ is the disparity gradient between $M_{pp'}$ and $M_{qq'}$ and DGL is the current disparity gradient limit.

Each zero-crossing q in N can be in one of the following situations:

1. Has already been matched with a zero-crossing q' in the right primal sketch. In this case, $DG\_TEST(p,q)$ is calculated easily using the above formula.

2. Is still unmatched. In this case, a matching zero-crossing q' is searched for in the right primal sketch such that $DG\_TEST(p,q) = 1$ If none can be found, $DG\_TEST(p,q)$ will be considered 0.

The values returned by $DG\_TEST(p,q)$ where $q \in N$ constitute the support that could be collected for $M_{pp'}$. Mathematically, the local support is defined as:

$$Support(M_{pp'}) = \sum_{q \in N} \frac{DG\_TEST(p,q)}{s(p,q)}$$

where $s(p,q)$ is the distance between (p_x, p_y) and (q_x, q_y).

The division by the distance between p and q is intended to give more importance to near supporters. This is because the probability that an incorrect match $M_{qq'}$ accidentally satisfying the DGL with $M_{pp'}$ is proportional to the distance between p and q.

The neighborhood size is dependent on the resolution level l. In the current implementation, a radius of 80 picture elements is used for the coarsest resolution. This value is divided by 2 after each level and thus becomes 10 in the fourth level.

4.4 Intensity Matching

Intensity matching is incorporated in our algorithm as follows. The fourth primal sketch pair, which is the finest resolution, is analyzed to find areas which are (almost) devoid of zero-crossings. Intensity matching is done in these areas using the original image pair. The disparity information already found for zero-crossings in the primal sketch is used to restrict the possible disparity range for intensity matching. The

rest of this section explains the details of how intensity matching is performed.

4.4.1 Primal sketch analysis

The analysis step is done in a row-wise manner. Each two consecutive matches found by the feature matching step are considered separately. A pair of matches $M_{pp'}$ and $M_{qq'}$ are considered consecutive if $p_y = q_y$ and there is no other reported match $M_{zz'}$ on the same row ($z_y = p_y$) such that $p_x < z_x < q_x$. Note that it follows from the ordering constraints that z'_x will not fall between p'_x and q'_x. Once the pair of matches $M_{pp'}$ and $M_{qq'}$ is located, the gap widths $q_x - p_x$ and $q'_x - p'_x$ are checked. If any of them exceeds a minimum gap width requirement MIN_GAP, the current pair of matches is skipped and the next one on the same row is considered. For matches which satisfy the MIN_GAP requirement, the left primal sketch current row is scanned from $p_x + 1$ to $q_x - 1$ and similarly, the right primal sketch current row is scanned from $p'_x + 1$ to $q'_x - 1$. For each scanned position (x, y), a square with a side length $SIDE$ is defined around (x, y). The number of zero-crossings in this square is computed. Then, the average of the number of zero-crossings is calculated for all squares. If this average falls above a certain threshold T, the area is considered dense of zero-crossings so the pair of matches is again removed from

further consideration. This threshold is defined as a fraction of the square area:

$$T = FRAC \cdot SIDE^2$$

If the test is passed, all the points from $p_x + 1$ to $q_x - 1$ in the left image are marked for later consideration for intensity matching with the points from $p'_x + 1$ to $q'_x - 1$ in the right image. Because the two strips of points are bounded by a pair of matches, it can be guaranteed that the ordering constraint will not be violated between the bounding matches and the outcoming matches of intensity matching.

In the current implementation, MIN_GAP is taken to be 8 picture elements, $SIDE$ is taken to be 9 picture elements, and $FRAC$ is 10%. All these values were determined empirically. They are meant to distinguish between areas dense of zero-crossings and sparse areas.

4.4.2 Matching intensity values

At this point, we have pairs of strips of points on which intensity matching will be performed. Each pair of strips is considered separately. The intensity matching process is very similar to that of feature matching.

A bipartite graph is constructed whose left and right part vertices correspond to points in the left and right strips, respectively. Each vertex p is augmented a pair

(p_x, p_y) corresponding to the point location in the image. The disparity D between a pair of vertices p and p' is given by:

$$D(p, p') = p'_x - p_x$$

For each pair of vertices p and p', a bounding rectangle with side length $SIDE$ is defined around each point. The strength of a match $Strength(M_{pp'})$ is calculated as the reciprocal of the sum of differences between corresponding intensity values in the two bounding rectangles:

$$Strength(M_{pp'}) = \frac{1}{1 + Diff(M_{pp'})}$$

where,

$$Diff(M_{pp'}) = \sum_{i=-SIDE/2}^{SIDE/2} \sum_{j=-SIDE/2}^{SIDE/2} |E_l(p_x + i, p_y + j) - E_r(p'_x + i, p'_y + j)|$$

and E_l and E_r are the original left and right image matrices.

An edge connects vertex p and p' if and only if the following requirements are met:

1. $Strength(M_{pp'}) > MIN\_STRENGTH$

   The $MIN\_STRENGTH$ threshold is used to remove those points which are very different from further consideration. The choice of $MIN\_STRENGTH$ is dependent on the intensity value range for the images. In the current implementation where pixels have a range of 256 gray-scale values, $MIN\_STRENGTH$

is taken to be $\frac{1}{10000}$ which means that the value for each pixel in one bounding rectangle can be different on average by an amount of approximately 100 from the value of the corresponding pixel in the other bounding rectangle. We have chosen this value large enough to allow a large number of possible matches to compete.

2. The disparity between p and p' is within an acceptable range ($Disp\_range_l$) of a projected disparity. This projected disparity is computed from the disparity map found for primal sketch with the finest resolution. The projected disparity is calculated exactly in the same manner as shown in feature based matching. Thus, the requirement is

$$PD(p) - Disp\_range_l \geq D(p,p') \geq PD(p) + Disp\_range_l$$

An edge between vertices p and p' is given a weight equal to $Strength(M_{pp'})$. As we did in feature matching, the edge with the highest weight can be selected right away as an acceptable match. The edge with the next highest weight is selected next; and so on. Vertices are marked as their edges are selected to guarantee that a vertex is not matched with more than one vertex. Edges connecting at least one marked vertex are skipped. As we did in feature matching, the ordering constraint is enforced by comparing the current match with the two already accepted adjacent matches (if any). If the ordering constraint is violated, the match is rejected. A special case is when the points in each strip are very much alike an thus matching

becomes not possible. This has been taken care of by skipping those edges whose weight is identical to the edge before or after.

Again, this is implemented by sorting down the edges based on their weights and then sequentially selecting them and marking the vertices as appropriate.

Chapter 5

Results and Analysis

Our stereo matcher was implemented in C language on a NeXT computer. The program was run and tested on DECstation Alpha running OSF/1 operating system. For evaluation purposes, our algorithm was tested on a wide range of stereo pairs. These stereo pairs were designed to test special aspects of the algorithm. The evaluation process of a stereo algorithm in general is not a straight forward process. The difficulty arises from the impracticality of quantitative accuracy measurement in the case of natural images. Qualitative measurements are almost always used in this case. Generated random-dot stereograms, however, permit quantitative measurement of accuracy because the correct depth information for every pixel is known beforehand. However, unlike natural images, random-dot stereograms are inherently

dense of features and thus will be not subjected to intensity matching by the algorithm. To test intensity matching, we introduced a different type of synthetic images which have plenty of sparse areas where intensity matching can be tested. Again, because these images are also synthetic, quantitative error measurements are also possible.

Another problem in performance evaluation is the absence of a standard measure of performance and benchmark test cases in the literature. This makes the process of comparing the performance of our algorithm with other algorithms more difficult.

The choice of error measurement method is important. We need to define a method which will clearly state how good a solution is. One alternative is computing the mean square error between the correct disparity values and the disparity values of the output. Although this method is commonly used in many applications, it is not appropriate for our testing purpose. To see why it is the case, consider two solutions one of which is perfect except for a single (in one point) but very large error in disparity. The second contains several hundreds of small errors such that the mean square error of the first solution is larger than that of the second solution. The first solution, however, is considered a better solution because it recovered the correct depth information for most parts of the scene whereas the second contains a large number of distortions. From this we can conclude that if the disparity value is far from the correct value, it does not matter how far. What matters is whether the

value is correct or not. Therefore, we have decided to measure the goodness of the solution as the percentage of correct disparity values (percentage of correct matches). However, because very small errors do not affect the quality of the solution by much, we also calculated the percentage of disparity values which are wrong by an amount of at most 1. As we mentioned, there is no standard way of error measurement in the literature; but this method closely resembles how evaluation is often done.

In the rest of this chapter, to aid the visualization of the solutions, disparity values are represented as colors. Spectrum range is used for convenience. Thus, a color for a point ranges from red to purple indicating the disparity value for that point. A red colored point indicates the largest disparity value and thus means that this point has the highest elevation (closest to the viewer). Purple points on the other hand are the furthest from the viewer.

5.1 Test Cases

All the stereograms used to test the algorithm fall in one of three categories:

- Random-dot stereograms.
- Synthetic images.

- Natural Images.

5.1.1 Random-dot stereograms

Random-dot stereograms were produced using a random-dot stereogram generator. The generator was designed to provide the flexibility of constructing stereograms of any chosen size with a large number of surfaces of different disparities (depths). The generator also produces a data file containing the correct disparity values for each pixel in the image as specified by the user. This file is used later for comparison with the results produced by the stereo matcher to give a quantitative error measurement.

We start by considering a regular random-dot stereogram consisting of a wedding cake built from four planar layers. Figure 5.1 shows the stereo pair as well as the solution found by the algorithm. The stereogram has a dot density of 50%. It can be clearly seen that four square surfaces of different depths were recovered. Only 2% of all matches were wrong by a value of more than 1; and it took 83.1 seconds to perform the whole matching process. Table 5.1 gives statistical measurements of error and time for this sample which is labeled Wedding 50 in the table.

The next test case was meant to test how the algorithm deals with depth sheers. Depth sheers (large surface discontinuities) are critical because they violate the

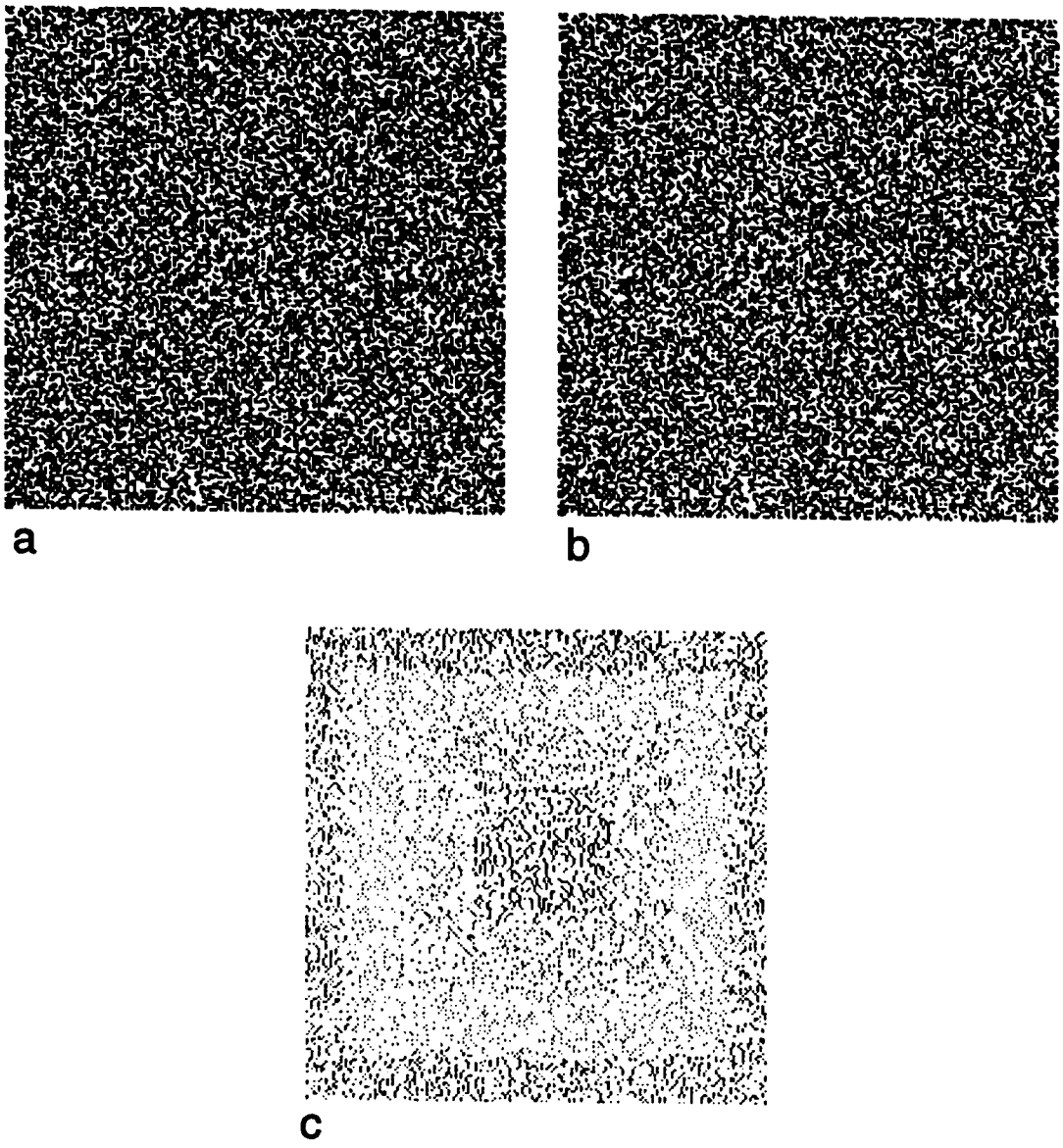


Figure 5.1: A stereogram portraying a wedding cake composed of for surfaces with a dot density of 50%. (a) left image. (b) right image. (c) result.

smoothness assumption made by the disparity gradient limit concept. The stereogram is composed of a horizontally oriented square wave with a peak-to-trough amplitude of 5 pixel. The disparity gradient between two matches which are close to each other but located on two surfaces of a different depth will definitely exceed the disparity gradient limit. The stereo pair and the solution are as in figure 5.2. Most of the matches were recovered correctly by the algorithm. The reason for this is that although the disparity gradient between matches close to an edge exceed the limit, no negative points are counted when calculating local support. Therefore, matches close to an edge gain the support of neighboring matches which are located on the same surface without being affected negatively by matches on the other surfaces. Of all matched zero-crossings, 4.8% were matched wrongly by a value of more than 1. The matching process took 55.4 seconds to complete. Exact figures are given in table 5.1 in the line labeled Sheers.

To test how the algorithm deals with sloping surfaces, a random-dot stereogram consisting of a horizontally oriented triangular wave with a peak-to-trough amplitude of 10 pixels was constructed. Most of the matches on the steep surfaces were recovered correctly. The stereogram and the solution are shown in figure 5.3. Of all matched zero-crossings, 5.5% were matched wrongly by a value of more than 1. The matching process took 92.4 seconds to complete. The results are shown in table 5.1 in the line labeled Sloping.

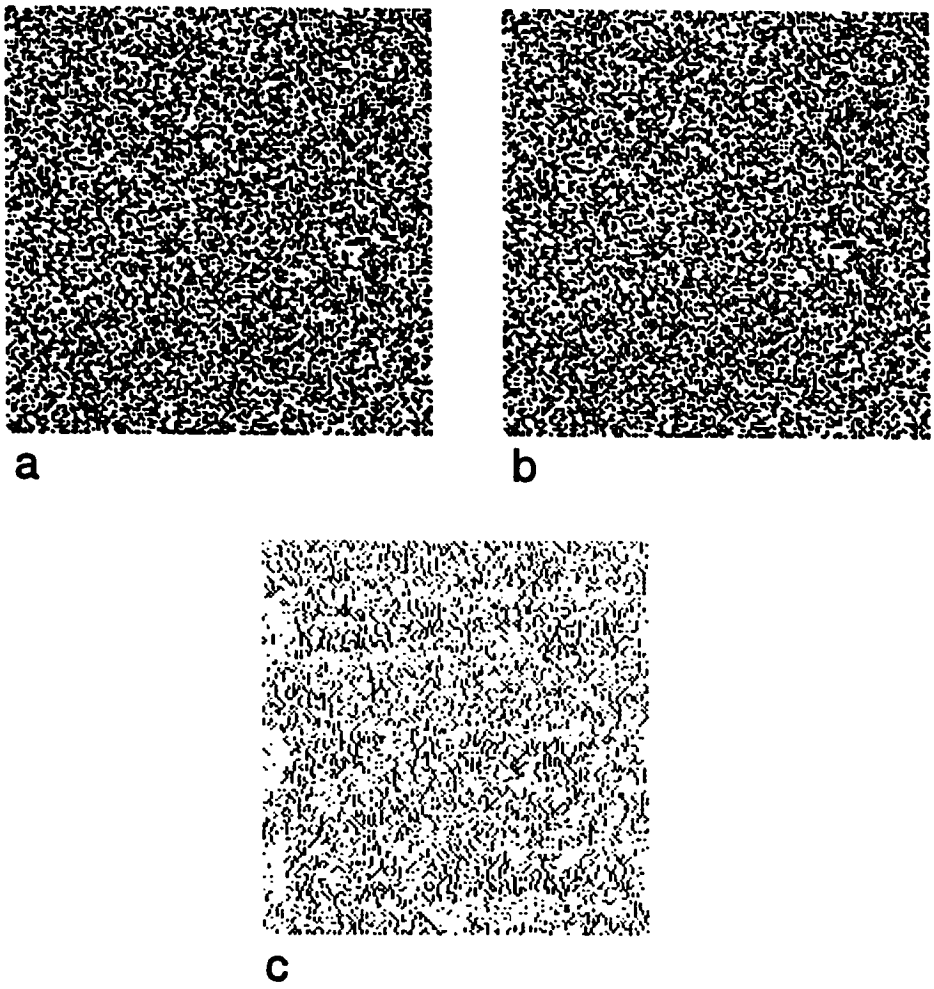


Figure 5.2: A stereogram portraying horizontal strips of surfaces with large depth discontinuities. (a) left image. (b) right image. (c) result.

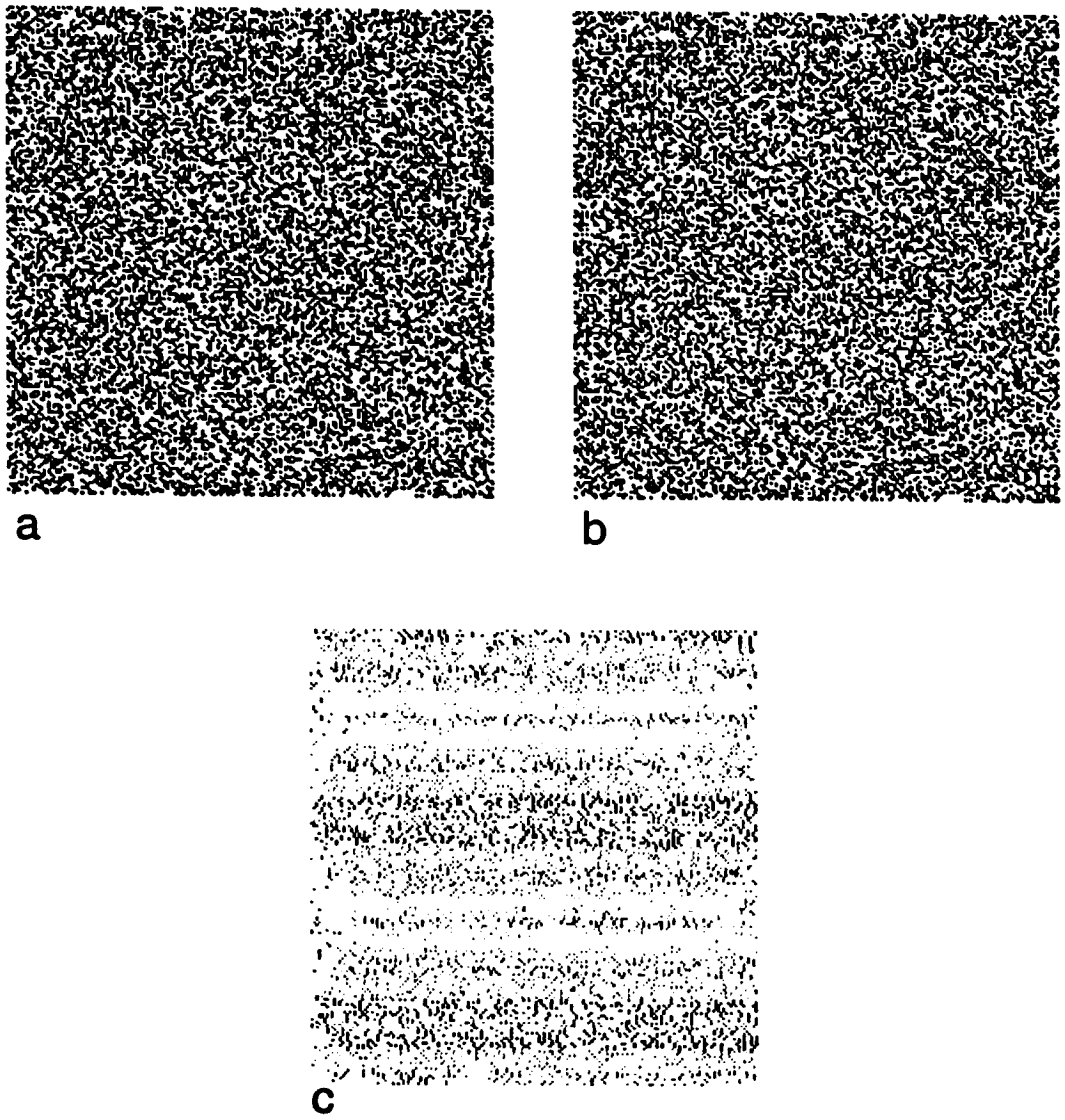


Figure 5.3: A stereogram portraying sloping surfaces. (a) left image. (b) right image. (c) result.

The algorithm was also tested on random-dot patterns with dot densities of 5%, 10%, and 25% all portraying a wedding cake. The percentage of wrong matches by a value of more than 1 was 2.7%, 2% and 1.6%, respectively; and the time taken to perform the matching was 69.9, 81.9, 86.2 seconds, respectively. Figure 5.4 shows the 5% stereogram and solution. The results for all three are shown in table 5.1 in the lines labeled Wedding 5, Wedding 10 and Wedding 25.

As can be seen from table 5.1, the performance of the algorithm on random-dot patterns is acceptable. In fact, it is comparable to, but not necessarily outperforming, the performance of many of the popular algorithms such as Grimson's implementation of the Marr-Poggio theory [Gri81] and the PMF algorithm [PMF85]. Regarding speed, our algorithm can be considered quite fast. For example, our algorithm ran faster by a magnitude of 10 compared to the PMF algorithm (but this figure may not be accurate if one takes care of differences in the speeds of the machines and the floating-point accelerators used.)

5.1.2 Other synthetic images

Another type of synthetic images was introduced to test the performance of intensity matching. Unlike random-dot stereograms, the primal sketches of these images should have areas with few zero-crossings in order for intensity matching to be

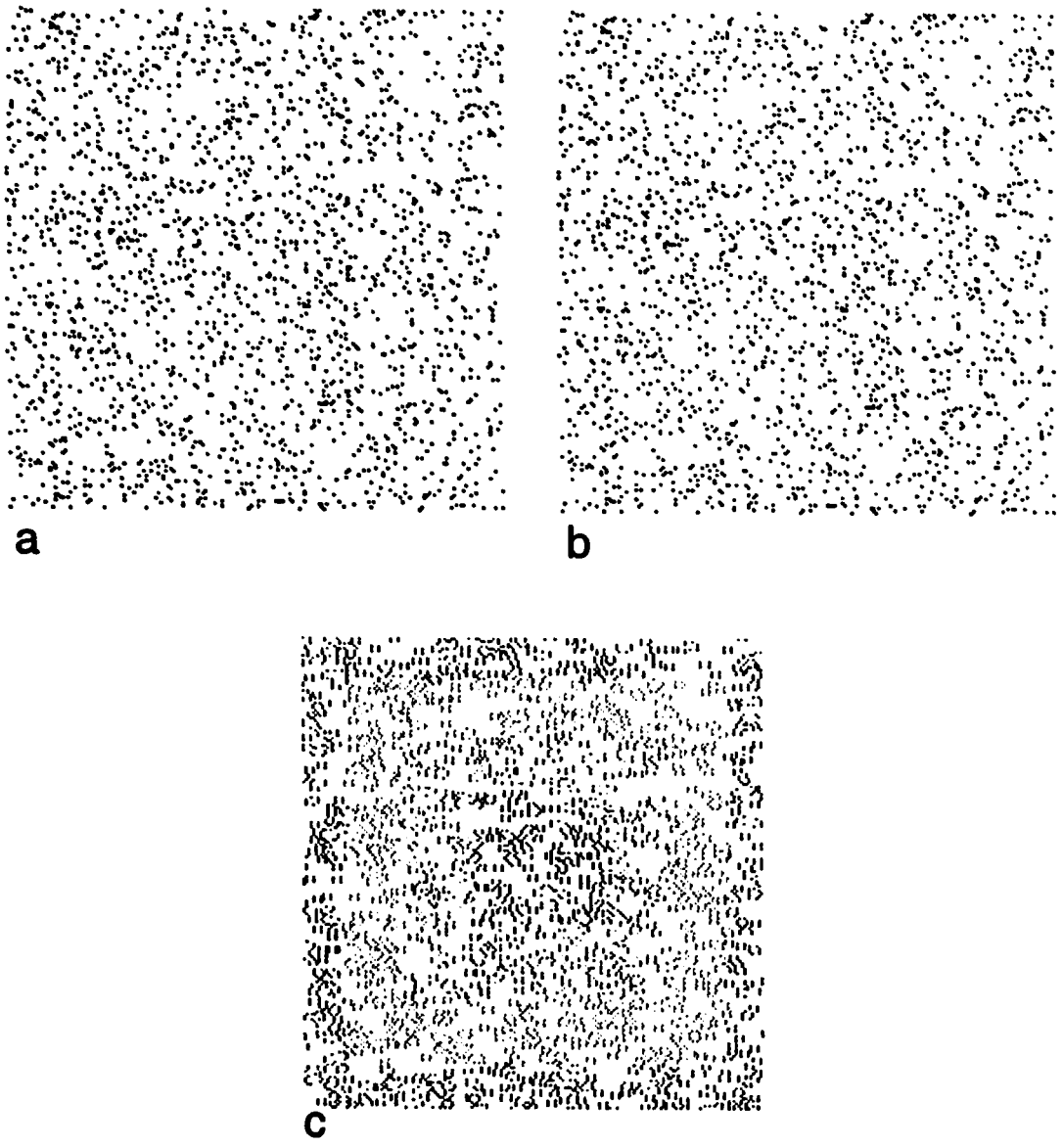


Figure 5.4: A stereogram portraying a wedding cake composed of for surfaces with a dot density of 5%. (a) left image. (b) right image. (c) result.

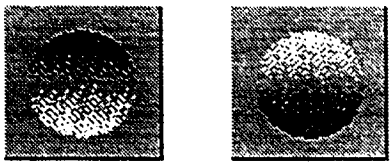


Figure 5.5: Depth reversal. The right image is an exact copy of the left one but flipped upside-down. The left image appears concave while the right one appears convex.

initiated. We have chosen two such images having this property. The first is of a sphere which does not have any texture on its surface. The sphere was rendered assuming a source of light from a given direction and then projected on two planes to give a stereo pair. The second is of a concave hemisphere which is rendered and projected in a similar manner.

It is interesting at this point to mention a phenomenon related to the shape-from-shading module in the human visual system. It has been noted that some concave surfaces, like the crates of the moon, are perceived convex when viewed upside-down. In fact, given any shaded image of a concave or a convex surface, the surface will be perceived concave if the bright side was at the bottom and convex if bright side was at the top. Figure 5.5 illustrates this phenomenon.

This phenomenon of this depth reversal was explained based on the fact that the earth has one sun which shines from above; and because the human visual system was made to cope with this environment, it gets tricked by such illusions. This explanation suggests that shape-from-shading is a high-level (top-down) process.

Therefore, stereo vision, being an early process, should not be affected by shape-from-shading. This was demonstrated by an experiment involving a stereo pair of a convex egg-crate with each member of the stereo pair appearing concave when viewed monocularly. When binocularly viewed, however, the egg-crate appeared convex [CJ90]. This shows that stereopsis dominates shape-from-shading in the human visual system.

The direction of light used for rendering the test images was in such a way that it would cause a depth reversal effect if each member of the stereo pair is viewed monocularly. This was done to test if our stereo matcher resembles the human visual system stereo matcher in not being influenced by the direction of light. Figures 5.6 and 5.7 show two stereo pairs of a convex and a concave hemisphere, respectively. The outputs clearly show that correct recovery of concavity was accomplished. In the case of the convex hemisphere, 11.6% of all matches were wrong by more than a value of 1; and it took 88.1 seconds to perform matching. In the case of the concave hemisphere, 6.1% of all matches were wrong by more than a value of 1; and it took 91.5 seconds to perform matching. Table 5.1 gives the statistical results in the lines lableled Convex and Concave.

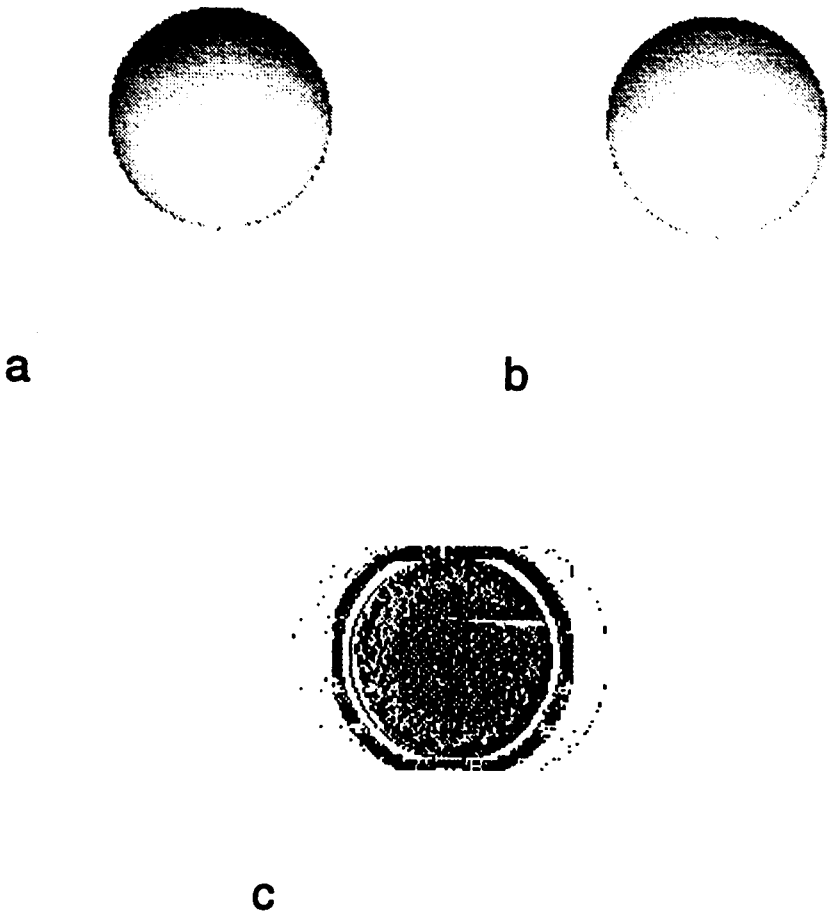
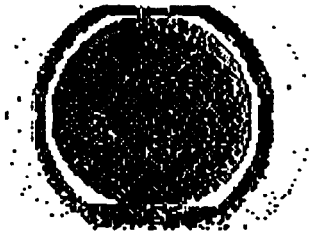


Figure 5.6: A stereogram portraying a sphere. (a) left image. (b) right image. (c) result.

a b



c

Figure 5.7: A stereogram portraying a concave hemisphere. (a) left image. (b) right image. (c) result.

5.1.3 Natural Images

A set of natural images was prepared using a Cannon camera connected to a Multimedia card. The images were all in gray-scale with each pixel having one value out of 256 possible values. The image sizes varied depending of the image being captured. The camera distance from the object being photographed was approximately 50cm. Because the camera had a fixed stand, instead of moving the camera, the object had to be shifted horizontally to capture the other member of the stereo pair. This introduces an error as will be explained below. A distance of 7cm was used between the two locations of the object.

The Hand example is shown in figure 5.8. As the results show, the gradual change in depth from the wrist towards the fingers was recovered correctly. The Biscuit stereopair shown in figure 5.9 involves two block objects one placed on top of the other. The Choke example in figure 5.10 has also two object but one of them is slanting. A cylinderical object is used in the Coke example as in figure 5.11. Finally, the Complex example in figure 5.12 involves an arrangement of several objects. Quialitative evaluation shows that the correct depth information for most of surfaces on the objects was recovered. Table 5.2 gives a summary of the execution time used to obtain these results.

Stereogram	Size	Total	Exact	1 Pixel	Wrong	%Wrong	Time (sec)
Wedding 50	203×203	9026	8606	242	178	2	83.1
Wedding 25	203×203	8905	8497	264	144	1.6	86.2
Wedding 10	203×203	8822	8475	175	172	2	81.9
Wedding 5	203×203	6345	6110	66	169	2.7	69.9
Sloping	193×193	7076	5875	814	387	5.5	92.5
Sheers	169×169	5826	5426	122	278	4.8	55.4
Convex	175×175	4361	1944	1910	507	11.6	88.1
Concave	175×175	5224	3110	1796	318	6.1	91.5

Table 5.1: A summary of the results for all the random-dot and synthetic images.

Stereogram	Size	Total	Time (sec)
Hand	180×250	6419	90
Biscuit	230×230	7608	102.6
Choke	230×220	5766	93.2
Coke	210×200	4622	66.3
Complex	250×220	6537	107.3

Table 5.2: Total matches and runtime for the natural images.

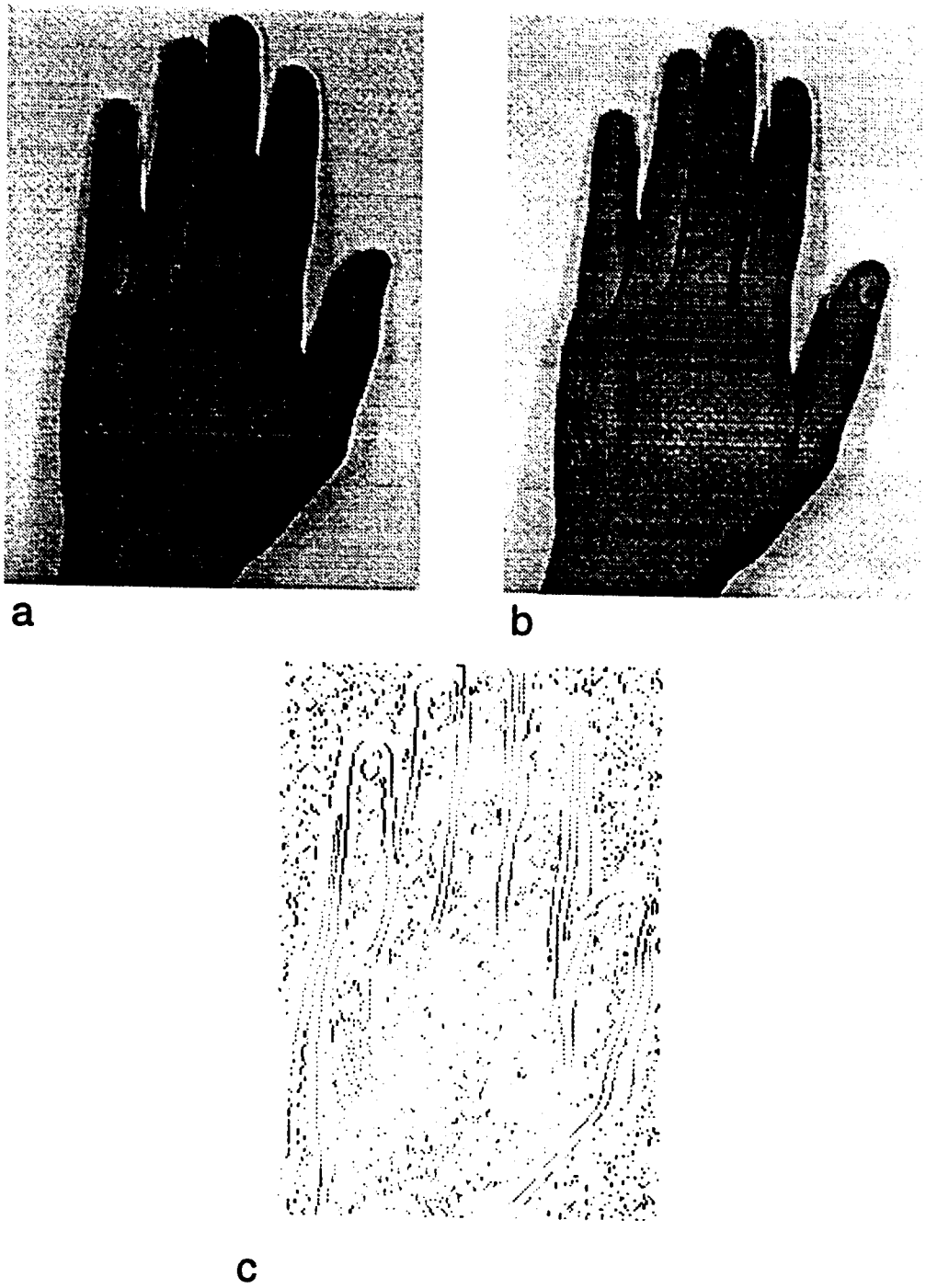


Figure 5.8: Hand (a) left image. (b) right image. (c) result.

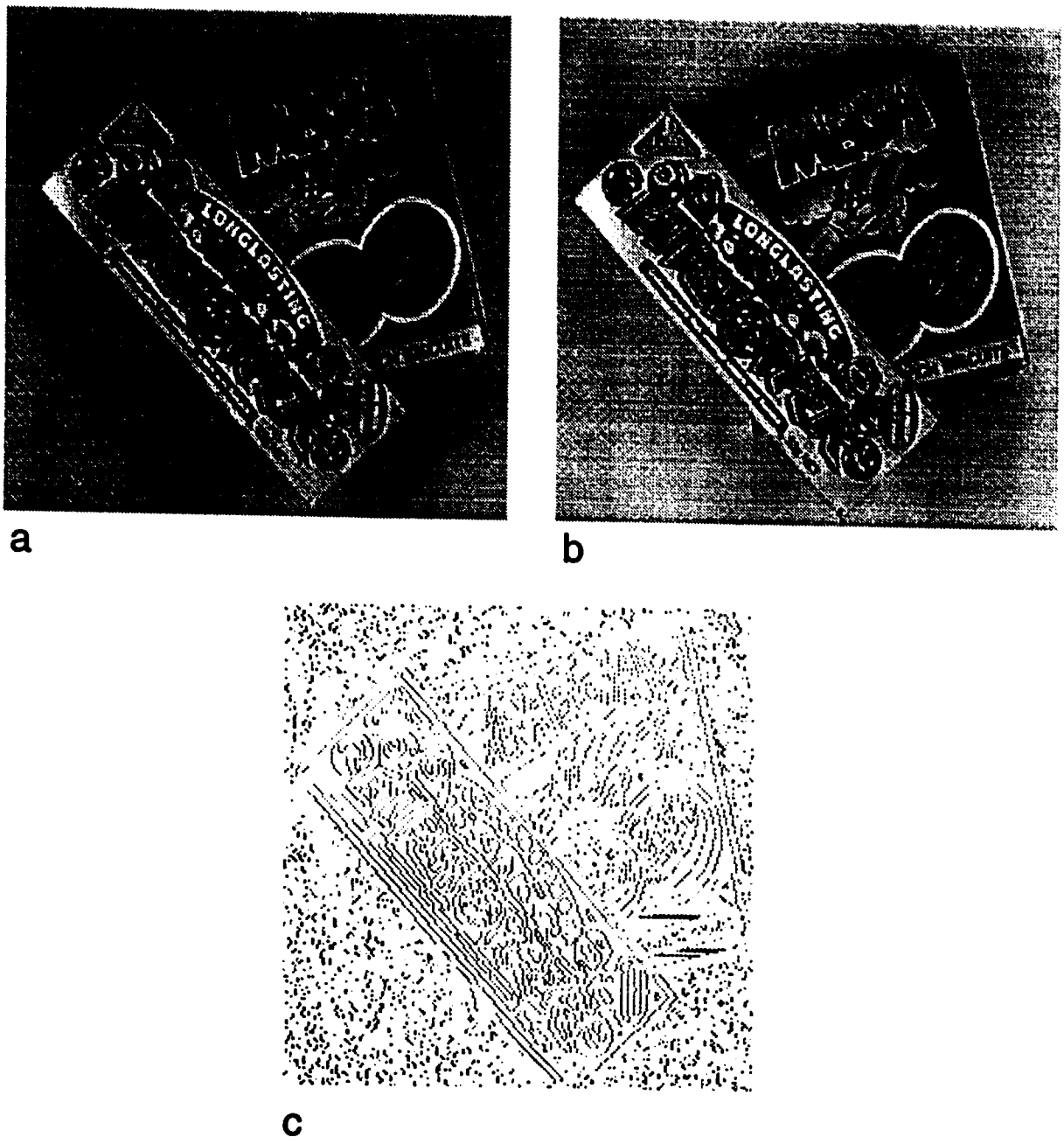


Figure 5.9: Biscuit (a) left image. (b) right image. (c) result.

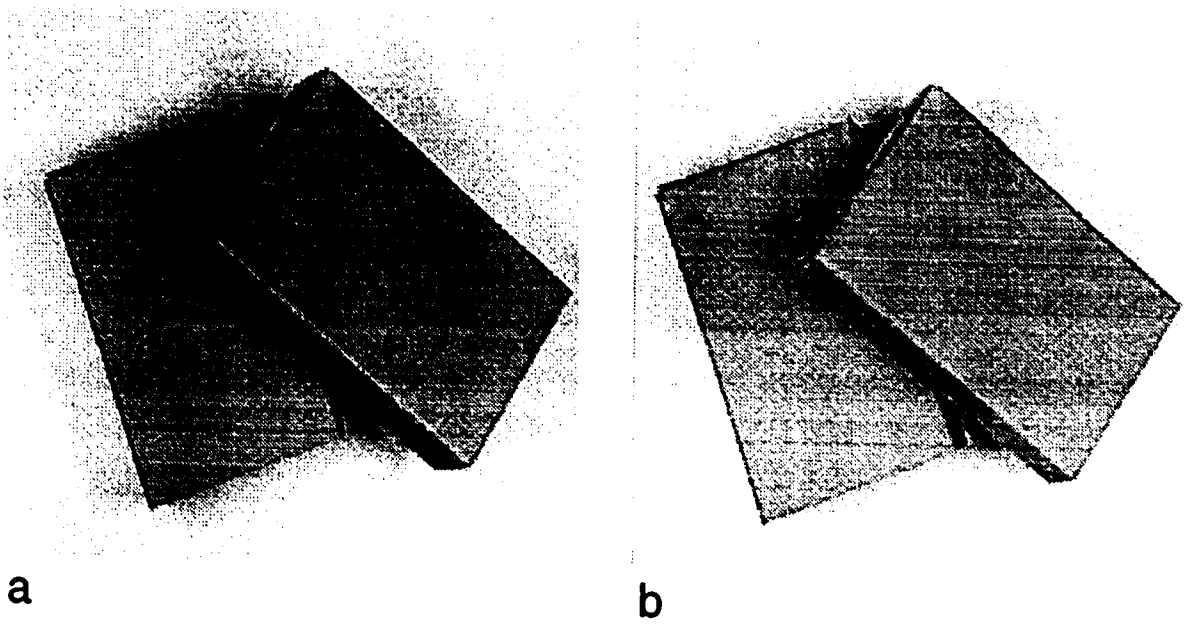


Figure 5.10: Choke (a) left image. (b) right image. (c) result.

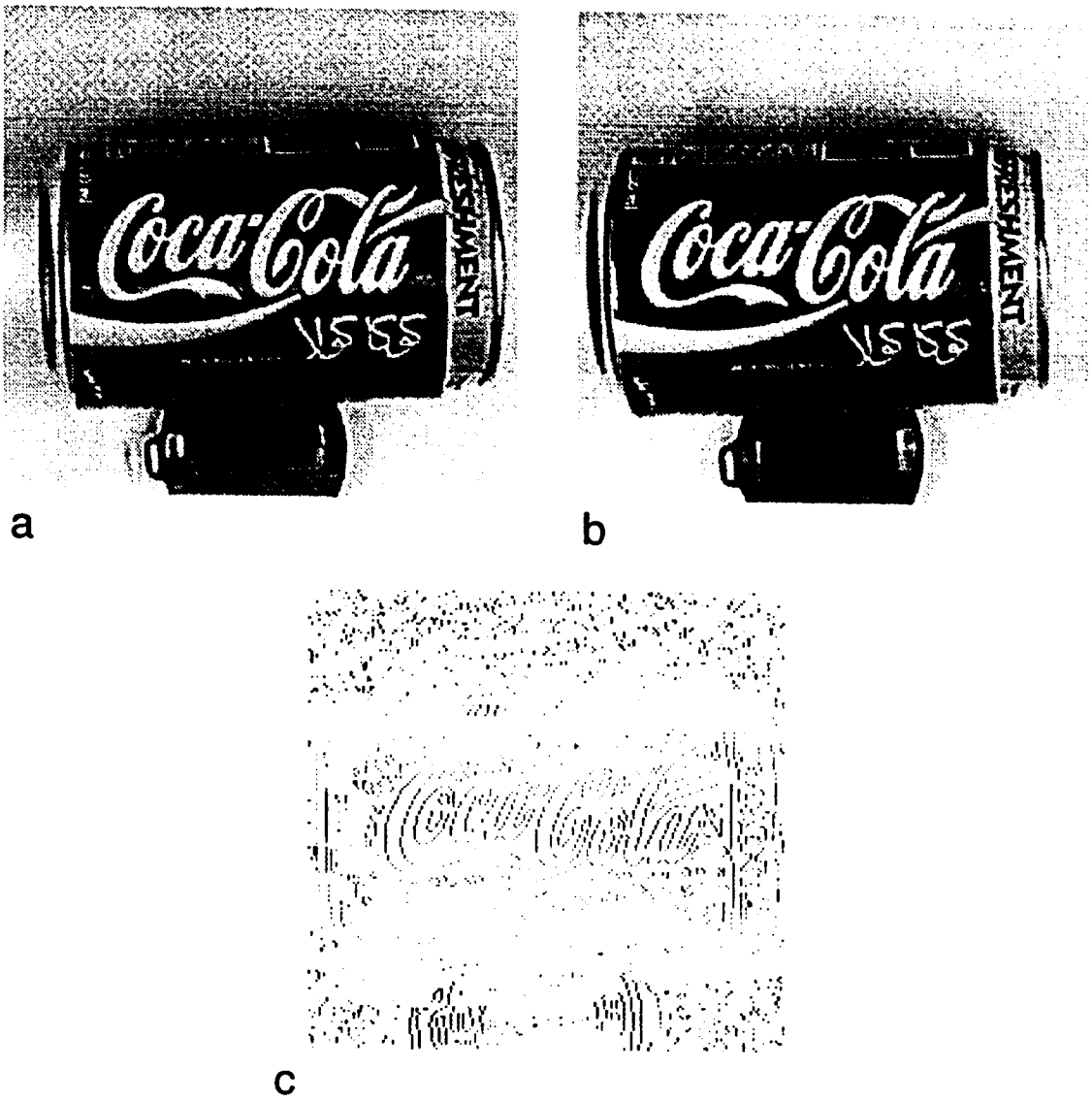


Figure 5.11: Coke (a) left image. (b) right image. (c) result.

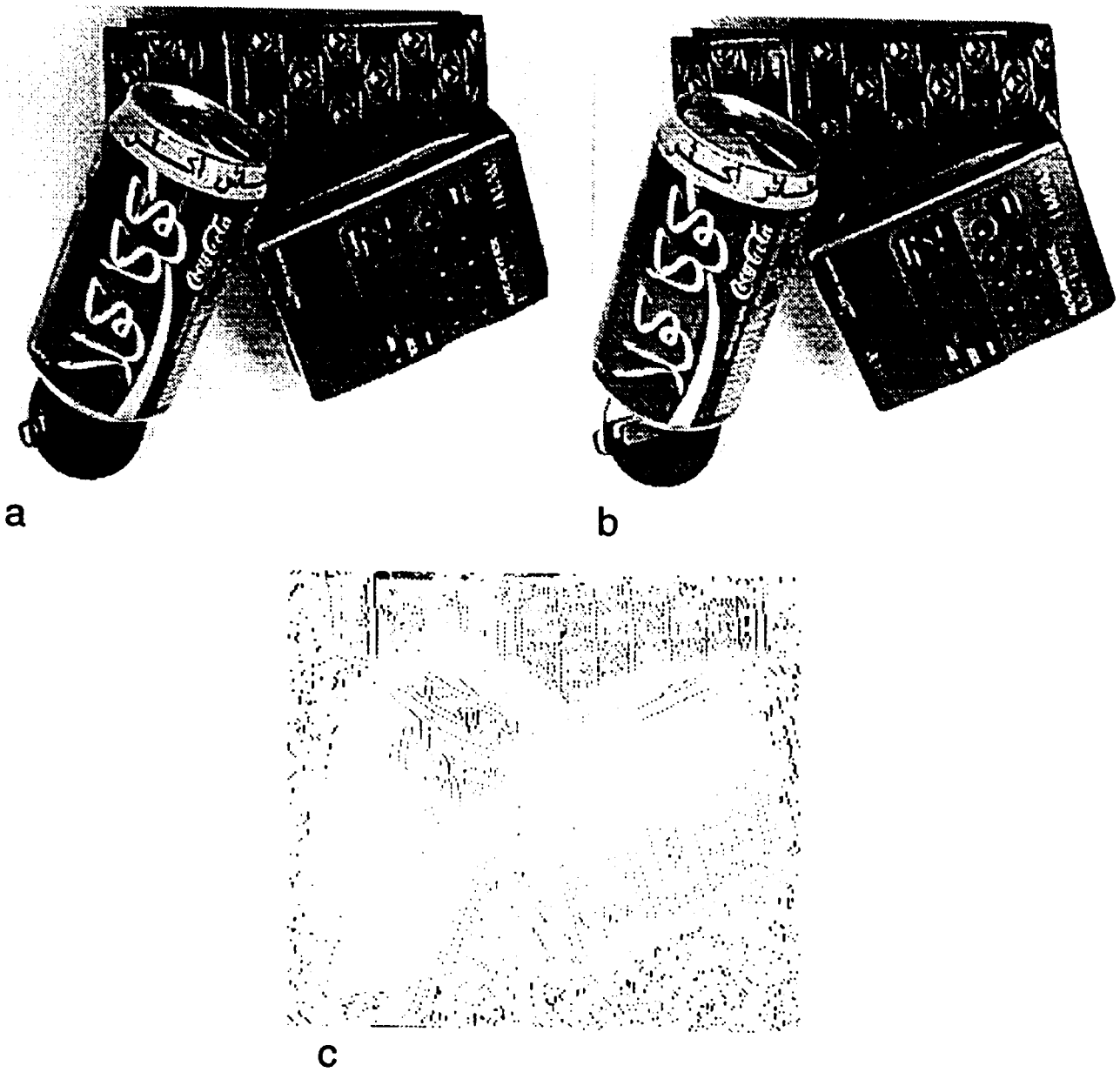


Figure 5.12: Complex (a) left image. (b) right image. (c) result.

5.2 Sources of Errors

Several sources of error were present in the case of natural images. Some of these errors, like camera noise, are inevitable. However, there were many other errors which affected the solution quality and could be eliminated.

As we mentioned before, the two stereo members were taken by manually shifting the object instead of moving the camera or using two cameras. Changing the position of an object, however, results in large change in the lighting conditions to which the object is exposed, especially if the source of light is close to the object (which was the case here as the camera stand has its own light source.) The largest effect of this error is on shadow boundaries which will no longer be located at the same distance from the object anymore. This difference in location reflects on the disparity value that will be given to the shadow boundaries and therefore, the depth of the shadow boundaries will not be the same as that of the background. This error would have been totally inexistent if two cameras were used or if the camera alone was moved instead of the objects.

The object shifting process also introduces vertical disparity which is assumed nonexistent by our algorithm (epipolar constraint.) A manual horizontal shift of an object will be expected to introduce a small vertical shift. However small this error, a degredation of the solution quality will be expected. A similar error results from the

camera itself. A straight line will appear bent if it is located away from the optical axes of the camera. Again, this would result in vertical disparity. Rectification techniques, which involve a series of transformations (translation, rotation, etc.,) are usually used to eliminate this kind of errors. Our algorithm does not include a rectification step, however, and thus these error will affect the solutions quality.

5.3 Speed and Complexity

Our program runs in two separate stages: edge detection and matching. The matching part is the main part of the algorithm. Edge detection involves a very expensive computational process which is convolution. Each member of the stereopair is convolved with the LoG operator four times to obtain the four primal sketches. The complexity of a convolution process is $O(n^2m^2)$ for an image of size $n \times n$ and a template of size $m \times m$. The template here is the LoG operator which vary in size depending on the filter width used (about 68×68 in the case of coarsest operator.) For an image of 230×230 pixels, convolution took about 7 minutes when run on DECstation Alpha. Convolution, however, is inherently parallel and there are many parallel algorithms developed for convolution. If an $n \times n$ mesh connected multicomputer is used, the above complexity can be cut down to $O(m^2)$ only [BJ90].

The complexity of the matching stage in our algorithm is dominated by the

process of collecting local support for candidate matches. Local support for each candidate match (composed of two zero-crossings) is calculated by considering all zero-crossings in the neighborhood. It is not possible to obtain a formula in terms of general parameters (like image size, etc.) describing the complexity of the matching process. This is because it is highly dependent of the nature of the image, clustering of the zero-crossings and other incalculable parameters. However, a crude estimation would be $O(k^2)$ where k is the number of zero-crossings in the finest primal sketch. However, if the number of zero-crossings is too low, intensity matching will take place causing a rise complexity. In all our expirements, the total time needed by the matcher never exceeded 2 minutes. For an image of 230 $\times$ 230 pixels, the matching process took about 107 seconds. This is considered fast when compared to execution times for other stereo algorithms in the literature. For example, the PMF algorithm took about 20 minutes to perform matching on a 256 $\times$ 256 image [BD91]. The algorithm by Baker and Binford [BB81] took about 19 minutes for an image of size 206 $\times$ 256 [OK85]. Grimson's revised implementation of Marr-Poggio theory [Gri85] takes in the order of 10 minutes to process an image of size 320 $\times$ 320.

Chapter 6

Conclusion and Future Work

In this work, we have introduced a stereo algorithm which combines feature matching as well as intensity matching in an attempt to recover as much detail of the three-dimensional scene as possible. Almost all stereo algorithms in the literature match either features or intensity values. Our stereo algorithm uses a coarse-to-fine strategy and runs in multiple passes at each resolution level. Feature matching is performed on zero-crossings in the primal sketches which are produced using Laplacian-of-Guassian filters of different sizes. Local matching attempts to match zero-crossings with similar signs and orientations. Global matching relies on the concept of disparity gradient limit through a relaxation process. Intensity values are matched as a final stage in areas lacking clusters of zero-crossings. The results

obtained at a certain point in the algorithm are used to guide the processing at later points. The algorithm was tested on random-dot stereograms and other synthetic images and good quantitative results were obtained. The algorithm was also tested on natural stereo pairs and gave good qualitative results. In all test cases, the total running time needed to perform the matching was small compared to the reported running times in the literature. This can be attributed to the coarse-to-fine strategy which propelled the search process.

Computational stereo research is motivated by two main directions. The first is concerned with having a better understanding and thus better emulation of the processing carried out in the human visual system. The second is application specific which is concerned with the need of having a good visual front-end which is able to obtain useful 3D information about the environment surrounding a robot. The lack of such front-end is one of the major obstacles encountered in automation of industrial tasks such as vehicle guidance. Stereo is a promising visual front-end because of its capability of extracting accurate depth information. However, the depth information in its raw form will not be useful and further processing is needed to obtain useful 3D information. Useful information include segmentation of the scene into smooth surface patches which describe different object, and recognition of the objects themselves. These are some of the possible ways towards which this work can be extended.

Another possible area of extending this work is trying to exploit the inherent parallelism involved in many parts of the algorithm. This is a necessity if a real-time system is to be designed. Another motivation for parillelization is to see if we can come close to the considerable speed of stereo matching done in the human visual system. Possible future implementation on transputer, neurocomputer, or other machines may be considered.

Appendix A

Matcher Listing

The program listing for the matcher part of our algorithm is given in this appendix. It assumes that edge detection and attribution has already been done. The matcher is executed by invoking the function *main_loop* which in turn proceeds to perform feature matching then intensity matching.

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

/* Tunable Parameters */
#define NO_OF_PASSES 4          /* Number of passes */
#define NO_OF_SUBPASSES 3       /* Number of repetitions per pass */
#define MIN_GAP 8               /* Gap size for intensity matching */
#define P_SIDE 9                /* Side length of the square area around a
                                 * pixel to be examined for zero-crossing
                                 * density */
#define D_SIDE 9                /* Side length of the square area around a
                                 * pixel defining the neighborhood */
#define P_MAX_FRACTION 0.1      /* Maximum tolerated percent of
                                 * zero-crossings for intensity matching */
#define MAX_ACCEPTABLE_DIFFERENCE 10000
/*
 * Maximum tolerated difference between two pixels' neighborhoods to be
 * considered for matching
 */

#define THETA_START 10          /* Starting value of delta theta */
#define THETA_SLACK 10          /* Delta theta slack */
#define THETA_STEP 10           /* Delta theta increment */

#define DG_START 0.6            /* Starting value for DGL */
#define DG_STEP 0.2             /* DGL increment */

#define SUPPORT_RADIUS 80       /* Starting support radius */
#define DISPARITY_RANGE 38      /* Starting search range */
#define DISPARITY_SEEK_RADIUS 60/* Starting radius for projected disparity
                                 * calculation in order to propagate
                                 * disparity information between resolutions */

/* Data Structures */

typedef struct                  /* structure for one image and its primal
                                 * sketches */
{
        int     color,          /* intensity value */
                correct;        /* correct disparity value (for testing) */
        struct
        {
                int     zc;     /* 1 if zero-crossing. 2 (decided by matcher)
                                 * if candidate for intensity matching */
                float   pos;    /* distance between zc exact position and
                                 * current pixel location [0,1) */
                float   sign;   /* positive/negative zc */
                int     dir;    /* local orientation of zc */
                int     surface;/* (decided by matcher) 1 if this zc/pixel
                                 * participated in an accepted match. 0
                                 * otherwise */
        }       conv[4];        /* 4 resoluion levels */
}     **map_ptr;                /* two dimensional array */

typedef struct                  /* structure for both images */
{
        int     width,          /* image width */
                height;         /* image height */
        struct
        {
                int     start,  /* xmin and ymin for each resolution level */
```

```
                        endx,   /* xmax for each resolution level */
                        endy;   /* ymax for each resolution level */
        }       conv[4];        /* 4 resolution levels */
        map_ptr map1,           /* left image and its primal sketches */
                map2;           /* right image and its primal sketches */
}       image_type;             /* structure parent */

/*
 * All values in the structures above except those marked as "decided by
 * matcher" are computed in the pre-matching steps (edge detection, etc.) and
 * thus assumed by the matcher to be ready. The reference to these structures
 * is through the external pointer "image" defined in external definitions
 * section below. The rest of the definitions below are specific to the
 * matcher
 */


typedef struct zc_list_record   /* structure for candidate zc list for
                                 * matching */
{
        float   xpos;           /* exact x position */
        int     col;            /* floor of x position */
        int     dir;
        int     sign;
        int     selected;       /* 1 if zc has already been matched */
        struct zc_list_record *next;
}       zc_list_rec, *zc_list_type;

typedef struct matches_record   /* structure for accepted matches in one row */
{
        float   disparity;      /* disparity */
        float   xpos1,          /* x pos in left image */
                xpos2;          /* x pos in right image */
        struct matches_record *next;    /* next match in the row */
}       matches_rec, *matches_type;

typedef struct row_of_matches_record    /* structure for matches in all rows */
{
        int     row_no;         /* y position */
        matches_type matches;
        struct row_of_matches_record *next,     /* the row below */
               *before;         /* the row above */
}       row_of_matches_rec, *row_of_matches_type;

typedef struct surface_record   /* structure for all matches found for one
                                 * resolution level */
{
        int     count;          /* number of matches */
        row_of_matches_type rows;       /* list of rows of matches */
}       surface_rec, *surface_type;


typedef struct candidate_match_record   /* structure for a candidate match */
{
        zc_list_type left,      /* zc in left image */
                right;          /* zc in right image */
        int     theta;          /* difference in orientation */
        float   disparity,
                support;        /* candidate match strength */
        struct candidate_match_record *next;    /* next candidate match */
}       candidate_match_rec, *candidate_match_type;
```

```
typedef struct                          /* structure for all candidate matches */
{
        int     count;                  /* number of candidate matches */
        candidate_match_type first;     /* their list head */
        candidate_match_type *sorted;   /* candidate matches sorted according
                                         * to match strength */
}       candidate_matches_rec, *candidate_matches_type;


typedef struct supporters_list_record   /* structure for list of possible
                                         * supporting matches */
{
        float   disparity,              /* disparity of supporter */
                distance,               /* distance (in left image) between zc in
                                         * question and supporter left zc */
                cycx,                   /* cyclopean x coordinate for supporter */
                cycy;                   /* cyclopean y coordinate for supporter */
        struct supporters_list_record *next;
}       supporters_list_rec, *supporters_list_type;


typedef struct zc_list_pixel_record     /* list of candidate pixels for
                                         * intensity matching */
{
        int     x;
        int     selected;               /* if it has already participated in a match */
        struct zc_list_pixel_record *next;
}       zc_list_pixel_rec, *zc_list_pixel_type;

typedef struct candidate_match_pixel_record     /* structure of a candidate
                                                 * match for intensity
                                                 * matching */
{
        zc_list_pixel_type left,/* pixel in left image */
                right;                  /* pixel in right image */
        float   disparity,
                support;                /* match strength */
        struct candidate_match_pixel_record *next;      /* next candidate match */
}       candidate_match_pixel_rec, *candidate_match_pixel_type;

typedef struct                          /* structure for all candidate matches for
                                         * intensity matching */
{
        int     count;                  /* number of candidate matches */
        candidate_match_pixel_type first;       /* list head */
        candidate_match_pixel_type *sorted;     /* sorted according to match
                                                 * strength */
}       candidate_matches_pixel_rec, *candidate_matches_pixel_type;


/* Function Prototypes */

void    main_loop(void);
void    initialize(surface_type[5]);
void    process_row(int, int, surface_type[5]);
void    kill_supporters(supporters_list_type);
void    kill_zc_list(zc_list_type);
zc_list_type add_to_zc_list(zc_list_type, float, int, int, int);
supporters_list_type get_supporters(float, int, row_of_matches_type);
```

```
supporters_list_type collect_supporters(row_of_matches_type, float, int, supporters_list_type);
float    get_approximate_disparity(zc_list_type, int, row_of_matches_type);
float    get_approximate_disparity_pixel(zc_list_pixel_type, int, row_of_matches_type);
void     row_disparity_approx(row_of_matches_type, float, int, float *, float *);
float    calculate_support(supporters_list_type, float, float, int);
row_of_matches_type locate_row(int, int, surface_type[5]);
void     add_new_match(candidate_match_type, int, row_of_matches_type, surface_type, int);
int      uncrossed(matches_type, float, float);
int      theta_difference(int, int);
int      within_acceptable_range(float, float);
void     sort_candidate_matches(candidate_matches_type);
int      support_comp(const void *, const void *);
float    compute_support(float, int, int, float);
void     analyze_image(surface_type);
void     analyze_pixels(int, int, int, int, int);
int      zc_sum(int, int, int, map_ptr);
void     process_intensity_row(int, surface_type[5]);
void     process_segment(int, int, int, int, int, surface_type[5]);
void     kill_zc_list_pixel(zc_list_pixel_type);
zc_list_pixel_type add_to_zc_list_pixel(zc_list_pixel_type, int);
float    difference(int, int, int);
void     sort_candidate_matches_pixel(candidate_matches_pixel_type);
int      support_comp_pixel(const void *, const void *);
void     add_new_match_pixel(candidate_match_pixel_type, int, row_of_matches_type, surface_type, int);
void     error_report(surface_type[5]);
void     error_report_pixel(surface_type surface[5]);


/* External Definitions */
extern void *reserve_mem(size_t);        /* dynamically allocate of memory */
extern void delete_mem(void *); /* release dynamically allocated memory */

extern image_type image;         /* A pointer to structures containing the
                                  * image information, primal sketches, etc. */
extern char fname[20];           /* file name of the image being processed */


/* Global Variables. Meanings are similar to the constant section. */
int      max_delta_theta,
         support_radius,
         disparity_seek_radius,
         disparity_range;
float    max_acceptable_dg;


/*
 * Initialization. Sets initial values for global varialbes.
 */
void     initialize(surface_type surface[5])
{
         int      i;

         support_radius = SUPPORT_RADIUS;
         disparity_seek_radius = DISPARITY_SEEK_RADIUS;
         disparity_range = DISPARITY_RANGE;

         for (i = 0; i < 5; i++)
         {
                  surface[i]->rows = NULL;
         }
}
```

```
/*
 * Starter function. After all image information is complete (image
 * structure), this function starts the matcher.
 */
void    main_loop(void)
{
        surface_rec surface_record[5];
        surface_type surface[5];/* 5 sets of matches. 1 for each of the
                                 * resolution levels. 1 for intensity
                                 * matching */

        int     j,
                resolution,
                pass,
                subpass;

        surface[0] = &surface_record[0];
        surface[1] = &surface_record[1];
        surface[2] = &surface_record[2];
        surface[3] = &surface_record[3];
        surface[4] = &surface_record[4];

        initialize(surface);

        /* feature matching */
        printf("Feature matching... ");

        for (resolution = 0; resolution < 4; resolution++)
        {
                max_acceptable_dg = DG_START; /* set initial DGL */
                max_delta_theta = THETA_START + THETA_SLACK; /* and initial
                                                              * delta_theta */

                printf("\nLevel %d: passes: ", resolution);
                for (pass = 0; pass < NO_OF_PASSES; pass++)
                {
                        printf("\n%d  ", pass);
                        for (subpass = 0; subpass < NO_OF_SUBPASSES; subpass++)
                        {
                                printf("%d ", subpass);
                                for (j = image.conv[resolution].start; j <=
                                     image.conv[resolution].endy; j++)      /* loop over all rows
                                                                             * for that primal
                                                                             * sketch */
                                        process_row(j, resolution, surface);        /* perform matching for
                                                                                     * row j */
                        }
                        max_delta_theta += THETA_STEP; /* update parameters
                                                        * after each pass */
                        max_acceptable_dg += DG_STEP;
                }
                disparity_range /= 2;       /* halve radius values after each
                                             * resolution */
                disparity_seek_radius /= 2;
                support_radius /= 2;
        }

        /* Intensity Matching */
        disparity_range = 15;
        disparity_seek_radius = 10;
```

```
        analyze_image(surface[3]);          /* image analysis for intensity
                                             * matching to find candidate strips
                                             * of pixels */

        printf("Intensity matching... ");
        for (subpass = 0; subpass < 3; subpass++)
        {
                printf("%d  ", subpass);
                for (j = image.conv[3].start; j <= image.conv[3].endy; j++)
                {
                        process_intensity_row(j, surface);          /* perform intensity
                                                                     * matching for row j */
                }
        }

        error_report(surface);
        error_report_pixel(surface);
}

/*
 * Matches zero-crossings in row j in the left image with those in the same
 * row in the right image. The matches are stored in surface[resolution]. The
 * process starts by forming two lists of zero-crossings in the two rows.
 * Then, a third list (candidate_matches list) is constructed whose elements
 * consist of pairs of pointers pointing to pairs of zero-crossings in each
 * of the first two lists. This list is sorted according to a strength value
 * and matches are accepted sequentially from the sorted list.
 */

void    process_row(int y, int resolution, surface_type surface[5])
{
        zc_list_type zc_list1,      /* zero-crossings list in the left image row */
                zc_list2,           /* zero-crossings list in the right image row */
                current_left_zc,
                current_right_zc;
        supporters_list_type supporters_list;       /* supporters list used to
                                                     * caculate the strength of a
                                                     * match */
        float   col1,
                col2,
                disparity,
                approx_disparity,           /* approximate disparity value
                                             * computed from the results at the
                                             * previous resolution */
                support;            /* match strength */
        int     i,
                delta_theta;        /* difference in orientation */
        row_of_matches_type row,/* current row of matches */
                parent_row;         /* row of matches in the previous resolution
                                     * level */
        candidate_matches_rec candidate_matches_record;
        candidate_matches_type candidate_matches;
        candidate_match_type candidate_match;

        candidate_matches = &candidate_matches_record;
        candidate_matches->count = 0;
        candidate_matches->first = NULL;

        zc_list1 = zc_list2 = NULL;
        col1 = image.conv[resolution].start;
```

```
col2 = image.conv[resolution].endx;

for (i = col1; i <= col2; i++)    /* scan the row from left to right */
{
        if (image.map1[i][y].conv[resolution].zc &&
            !image.map1[i][y].conv[resolution].surface)        /* if there is a
                                                                * zero-crossing which
                                                                * is not matched                    380
                                                                * already, add it to
                                                                * the list */
                zc_list1 = add_to_zc_list(zc_list1, i + image.map1[i][y].conv[resolution].pos, i,
                                image.map1[i][y].conv[resolution].dir,
                                image.map1[i][y].conv[resolution].sign);
        if (image.map2[i][y].conv[resolution].zc &&
            !image.map2[i][y].conv[resolution].surface)        /* same as above but for
                                                                * the right primal
                                                                * sketch */
                zc_list2 = add_to_zc_list(zc_list2, i + image.map2[i][y].conv[resolution].pos, i,    390
                                image.map2[i][y].conv[resolution].dir,
                                image.map2[i][y].conv[resolution].sign);
}


row = locate_row(y, resolution, surface);                  /* find the row of
                                                            * matches which is
                                                            * closest to y from
                                                            * above */
if (resolution > 0)                                                                               400
        parent_row = locate_row(y, resolution - 1, surface);          /* find the row of
                                                                       * matches in the
                                                                       * previous resolution
                                                                       * which is closest to y
                                                                       * from above */
else
        parent_row = NULL;        /* there is no previous resolution
                                   * level */

current_left_zc = zc_list1;                                                                       410
while (current_left_zc != NULL)   /* traverse the list of left row
                                   * zero-crossings */
{
        /*
         * calculate the projected disparity from the previous
         * resolution level
         */
        approx_disparity = get_approximate_disparity(current_left_zc, y, parent_row);
        /*
         * make a list of supporters (matches) whose left                                           420
         * zero-crossing member is within the support radius of the
         * current left zc
         */
        supporters_list = get_supporters(current_left_zc->xpos, y, row);
        current_right_zc = zc_list2;
        while (current_right_zc != NULL)          /* traverse the list of
                                                   * right row zc's */
        {
                if (current_right_zc->sign == current_left_zc->sign)          /* if identical signs */
                {                                                                                 430
                        /* calculate orientation difference */
                        delta_theta = theta_difference(current_left_zc->dir,
                                                       current_right_zc->dir);
                        if (delta_theta <= max_delta_theta)          /* if below threshold */
```

```
                {
                        disparity = current_right_zc->xpos -
                                current_left_zc->xpos;
                        /*
                         * if disparity agrees with that
                         * computed from previous level                    440
                         */
                        if (within_acceptable_range(disparity,
                                        approx_disparity))
                        {
                                /*
                                 * compute match strength
                                 * using the list of
                                 * supporters
                                 */
                                support = calculate_support                 450
                                        (supporters_list, disparity,
                                  current_left_zc->xpos, y);
                                /*
                                 * compute extra strength
                                 * from support by possible
                                 * future matches
                                 */
                                support += compute_support
                                        (current_left_zc->xpos, y,
                                     resolution, disparity);                460
                                /*
                                 * register match information
                                 * for later consideration
                                 */
                                candidate_match =
                                        (candidate_match_type)
                                        reserve_mem(sizeof(candidate_match_rec));
                                candidate_match->left = current_left_zc;
                                candidate_match->right = current_right_zc;
                                candidate_match->support = support;         470
                                candidate_match->disparity =
                                        disparity;
                                candidate_match->theta =
                                        delta_theta;
                                candidate_match->next =
                                        candidate_matches->first;
                                candidate_matches->first =
                                        candidate_match;
                                candidate_matches->count++;
                        }                                                   480
                }
        }
        current_right_zc = current_right_zc->next;
    }
    kill_supporters(supporters_list);           /* free supporters list
                                                 * for this zc (no
                                                 * longer needed) */
    current_left_zc = current_left_zc->next;
}
                                                                            490
sort_candidate_matches(candidate_matches);      /* sort descendingly
                                                 * according to strength */

for (i = 0; i < candidate_matches->count; i++)  /* traverse the sorted
                                                 * list */
{
```

```
                candidate_match = candidate_matches->sorted[i];


                if (!candidate_match->left->selected && !candidate_match->right->selected)
                        /*
                         * if the two match members have not participated in
                         * an accepted match yet
                         */
                {
                        if (candidate_match->theta <= max_delta_theta - THETA_SLACK)
                                /*
                                 * make sure it is withing delta theta
                                 * without the slack
                                 */
                        {
                                add_new_match(candidate_match, y, row,
                                        surface[resolution], resolution);
                                /* accept match */
                                if (row == NULL || row->row_no != y)
                                        row = locate_row(y, resolution,
                                                surface);
                        }
                }
                candidate_match->left->selected = 1;    /* mark zc's as selected */
                candidate_match->right->selected = 1;
                delete_mem(candidate_match);
        }

        if (candidate_matches->count > 0)
                delete_mem(candidate_matches->sorted);

        kill_zc_list(zc_list1);         /* free memory */
        kill_zc_list(zc_list2);
}

/*
 * free memory from list of supporters
 */
void    kill_supporters(supporters_list_type supporters_list)
{
        void    *tmp;

        while (supporters_list != NULL)
        {
                tmp = supporters_list;
                supporters_list = supporters_list->next;
                delete_mem(tmp);
        }
}

/*
 * free memory from zero-crossing list
 */
void    kill_zc_list(zc_list_type zc_list)
{
        void    *tmp;

        while (zc_list != NULL)
        {
                tmp = zc_list;
                zc_list = zc_list->next;
```

```
                delete_mem(tmp);
        }
}

/*
 * add a new member to the list of zero-crossings.
 */
zc_list_type add_to_zc_list(zc_list_type zc_list, float xpos, int col, int dir, int sign)
{
        zc_list_type new_zc;

        new_zc = (zc_list_type) reserve_mem(sizeof(zc_list_rec));
        new_zc->xpos = xpos;
        new_zc->col = col;
        new_zc->dir = dir;
        new_zc->sign = sign;
        new_zc->selected = 0;
        new_zc->next = zc_list;

        return new_zc;
}

/*
 * Construct the supporting matches list. They are the matches whose left
 * zero-crossing member falls within support_radius. This function passes
 * rows of matches to collect_supporters which in turn extracts the
 * supporting matches
 */
supporters_list_type get_supporters(float xpos, int y, row_of_matches_type row)
{
        supporters_list_type supporters_list;
        row_of_matches_type current_row;


        supporters_list = NULL;
        current_row = row;
        /* start from the current row and go upward */
        while (current_row != NULL && current_row->row_no >= y - support_radius)
        {
                /* get supporters in current_row */
                supporters_list = collect_supporters(current_row, xpos, y, supporters_list);
                current_row = current_row->before;
        }

        if (row)
                current_row = row->next;
        else
                current_row = NULL;

        /* start from the current row and go downward */
        while (current_row != NULL && current_row->row_no <= y + support_radius)
        {
                /* get supporters in current_row */
                supporters_list = collect_supporters(current_row, xpos, y, supporters_list);
                current_row = current_row->next;
        }

        return supporters_list;
}

/*
```

```
 * For the row under consideration, locates all zero-crossings which are
 * already matched and whose position is within support_radius neighborhood.
 */
supporters_list_type collect_supporters(row_of_matches_type current_row, float xpos, int ypos,
                                        supporters_list_type supporters_list)
{
        int     y;
        float   xdiff2,
                ydiff2;
        supporters_list_type supporter;                                          630
        matches_type match;

        y = current_row->row_no;
        ydiff2 = (float)(y - ypos) * (y - ypos);
        match = current_row->matches;
        while (match != NULL && match->xpos1 < xpos - support_radius)
                /*
                 * locate the first match in the row which falls within the
                 * neighborhood
                 */                                                              640
                match = match->next;

        while (match != NULL && match->xpos1 <= xpos + support_radius)
        {
                xdiff2 = (xpos - match->xpos1);
                xdiff2 = xdiff2 * xdiff2;
                if (ydiff2 + xdiff2 <= support_radius * support_radius)
                {
                        supporter = (supporters_list_type) reserve_mem(sizeof(supporters_list_rec));
                        supporter->distance = sqrt(ydiff2 + xdiff2);        /* distance between the    650
                                                                             * two zc's in the left
                                                                             * image */
                        supporter->cycx = (match->xpos1 + match->xpos2) / 2; /* the cyclopean
                                                                              * x-coordinate for the
                                                                              * match */
                        supporter->cycy = y;    /* the cyclopean y-coordinate
                                                 * for the match */
                        supporter->disparity = match->disparity;
                        supporter->next = supporters_list;
                        supporters_list = supporter;                             660
                }
                match = match->next;
        }
        return supporters_list;
}


/*
 * Compute a projected disparity value for the zc in question using the
 * disparity information for zc's in the previous resolution level. The     670
 * function passes rows of matches to row_disparity_approx which in turn does
 * the calculation.
 */
float   get_approximate_disparity(zc_list_type zc, int y, row_of_matches_type row)
{
        float   num,            /* numerator of the averaging formula */
                dom,            /* denomenator */
                approx;
        row_of_matches_type current_row;

                                                                                 680
        num = dom = 0;
        current_row = row;
```

```
        while (current_row != NULL && current_row->row_no >= y - disparity_seek_radius)
                /* send the rows above the current */
        {
                row_disparity_approx(current_row, zc->xpos, y, &num, &dom);
                current_row = current_row->before;
        }

        if (row)                                                                    690
                current_row = row->next;
        else
                current_row = NULL;

        while (current_row != NULL && current_row->row_no <= y + disparity_seek_radius)
                /* send the rows below the current */
        {
                row_disparity_approx(current_row, zc->xpos, y, &num, &dom);
                current_row = current_row->next;
        }                                                                           700

        if (dom > 0)
                approx = num / dom;
        else
                approx = 0;


        return approx;
}

                                                                                    710
/*
 * Compute a weigted average of disparity values for matches in a single row
 * which fall within a neighborhood of radius disparity_seek_radius from the
 * zero-crossing in question.
 */
void    row_disparity_approx(row_of_matches_type current_row, float xpos, int ypos,
                                        float *num, float *dom)
{
        int     y;
        float   xdiff2,                                                             720
                ydiff2,
                inverted_distance,
                inverted_distance_sum,
                weighted_sum;
        matches_type match;

        inverted_distance_sum = weighted_sum = 0;
        y = current_row->row_no;
        ydiff2 = (float)(y - ypos) * (y - ypos);
        match = current_row->matches;                                               730
        while (match != NULL && match->xpos1 < xpos - disparity_seek_radius)
                match = match->next;

        while (match != NULL && match->xpos1 <= xpos + disparity_seek_radius)
        {
                xdiff2 = (xpos - match->xpos1);
                xdiff2 = xdiff2 * xdiff2;
                if (ydiff2 + xdiff2 <= disparity_seek_radius * disparity_seek_radius)
                {
                        inverted_distance = 1 / (sqrt(ydiff2 + xdiff2) + 1);        740
                        inverted_distance_sum += inverted_distance;
                        /*
                         * sum reciprocals of distances between the two zc's.
                         * the reciprocal of the distance between the two
```

```
                        * zc's is the weight given to the current disparity
                        * value
                        */
                       weighted_sum += match->disparity * inverted_distance;
                }
                match = match->next;                                          750
        }

        *num = *num + weighted_sum;   /* update sums */
        *dom = *dom + inverted_distance_sum;
}

/*
 * Calculate match strength as being the degree by which the match satisfies
 * the DGL with the surrounding supporters.
 */                                                                           760
float   calculate_support(supporters_list_type supporters_list, float disparity, float xpos, int ypos)
{
        float   support;
        supporters_list_type supporter;
        float   delta_disparity,
                disparity_gradient;
        float   cyc_distx,
                cyc_disty,
                cyc_dist,
                cycx;                                                         770

        supporter = supporters_list;
        support = 0;
        cycx = xpos + disparity / 2;      /* cyclopean x-coordinate for the
                                           * candidate match */
        while (supporter != NULL)
        {
                delta_disparity = disparity - supporter->disparity;
                cyc_distx = supporter->cycx - cycx;
                cyc_disty = supporter->cycy - ypos;                           780
                cyc_dist = sqrt(cyc_distx * cyc_distx + cyc_disty * cyc_disty);        /* cyclopean distance */

                if (delta_disparity == 0 || cyc_dist != 0)
                {
                        if (delta_disparity == 0)
                                disparity_gradient = 0;
                        else
                                disparity_gradient = fabs(delta_disparity / cyc_dist);
                        if (disparity_gradient <= max_acceptable_dg)        /* satisfy DGL */
                                support += 1 / (supporter->distance);         790
                }
                supporter = supporter->next;
        }

        return support;
}

/*
 * Locate the row of matches whose row number is less than or equal to y and
 * as close to y as possible.                                                 800
 */
row_of_matches_type locate_row(int y, int resolution, surface_type surface[5])
{
        row_of_matches_type row_before,
                current_row;
```

```
        row_before = current_row = surface[resolution]->rows;
        while (current_row != NULL && current_row->row_no < y)
        {
                row_before = current_row;                                    810
                current_row = current_row->next;
        }
        if (current_row == NULL)
                current_row = row_before;

        return current_row;
}

/*
 * Add match to the matches list in the appropriate row and position.      820
 */
void    add_new_match(candidate_match_type candidate_match, int y,
              row_of_matches_type row, surface_type surface, int resolution)
{
        row_of_matches_type before,
                new_row;
        matches_type current_match,
                match_before,
                new_match;
                                                                             830
        if (row == NULL || row->row_no != y)  /* if row doesn't exist, make
                                               * one */
        {
                before = row;
                if (row != NULL)
                {
                        if (row->row_no < y)
                                row = row->next;
                        else
                                before = row->before;                        840
                }
                new_row = (row_of_matches_type) reserve_mem(sizeof(row_of_matches_rec));
                new_row->row_no = y;
                new_row->matches = NULL;
                new_row->before = before;
                new_row->next = row;
                if (row != NULL)
                        row->before = new_row;
                if (before != NULL)
                        before->next = new_row;                              850
                else
                        surface->rows = new_row;

                row = new_row;
        }
        current_match = row->matches;
        match_before = NULL;
        /* locate correct match position within current row */
        while (current_match != NULL && current_match->xpos1 < candidate_match->left->xpos)
        {                                                                    860
                match_before = current_match;
                current_match = current_match->next;
        }

        /* enforce the ordering constraint */
        if (uncrossed(match_before, candidate_match->left->xpos,
                        candidate_match->right->xpos) &&
            uncrossed(current_match, candidate_match->left->xpos,
```

```
                candidate_match->right->xpos))
        {                                                                    870
                /* add match to the list */
                new_match = (matches_type) reserve_mem(sizeof(matches_rec));
                new_match->xpos1 = candidate_match->left->xpos;
                new_match->xpos2 = candidate_match->right->xpos;
                new_match->disparity = candidate_match->disparity;
                new_match->next = current_match;
                if (match_before)
                        match_before->next = new_match;
                else
                        row->matches = new_match;                            880


                surface->count++;

                /* mark zero-crossing as matched */
                image.map1[candidate_match->left->col][y].conv[resolution].surface = 1;
                image.map2[candidate_match->right->col][y].conv[resolution].surface = 1;
        }
}                                                                            890

/*
 * return true if order is preserved between match and the match composed of
 * (xpos1,xpos2)
 */
int     uncrossed(matches_type match, float xpos1, float xpos2)
{
        if      (match)
        {
                if      ((match->xpos1 >= xpos1 && match->xpos2 >= xpos2) ||
                         (match->xpos1 <= xpos1 && match->xpos2 <= xpos2))   900
                                return 1;
                return 0;
        }
        return 1;
}

/*
 * Compute difference in orientation between two angles. If any (or both) is
 * 90, the difference will be such that it is considered acceptable in the
 * last pass                                                                 910
 */
int     theta_difference(int theta1, int theta2)
{
        int     diff;

        if (theta1 == 90 || theta2 == 90)
                return THETA_START + THETA_STEP * (NO_OF_PASSES - 1);
        diff = abs(theta1 - theta2);
        return diff;
}                                                                           920

/*
 * Check if the difference between recovered disparity and projected
 * disparity is within the permitted range.
 */
int     within_acceptable_range(float disparity, float approx_disparity)
{
        if      (fabs(disparity - approx_disparity) <= disparity_range)
                        return 1;
        return 0;                                                           930
```

```
}

/*
 * Sort candidate matches mased on the support (strength) field
 */
void    sort_candidate_matches(candidate_matches_type candidate_matches)
{
        candidate_match_type current_match;
        int     i;

        if (candidate_matches->count)
                candidate_matches->sorted = (candidate_match_type *) reserve_mem(
                    candidate_matches->count * sizeof(candidate_match_type));
        else
                candidate_matches->sorted = NULL;

        current_match = candidate_matches->first;
        i = 0;
        while (current_match != NULL)
        {
                candidate_matches->sorted[i] = current_match;
                i++;
                current_match = current_match->next;
        }

        if (candidate_matches->count)
                qsort((void *)candidate_matches->sorted, candidate_matches->count,
                        sizeof(candidate_match_type), support_comp);

}

/*
 * Used by the preceeding function.
 */
int     support_comp(const void *a, const void *b)
{
        if      ((*(candidate_match_type *) a)->support >
                        (*(candidate_match_type *) b)->support)
                        return -1;
        if ((*(candidate_match_type *) a)->support <
            (*(candidate_match_type *) b)->support)
                return 1;
        return 0;
}


/*
 * Compute support which would be given to the candidate match by possible
 * future matches
 */
float   compute_support(float xpos, int y, int resolution, float disparity)
{
        int     i,
                j,
                k,
                col1,
                col2,
                row1,
                row2,
                search_col1,
                search_col2,
                sign1,
```

```
        sign2,
        found;
float   support,
        distance,
        xdiff2,
        ydiff2;
float   xdiff,
        dg2,                                                        1000
        val1,
        val2,
        first_part,
        sqrt_part,
        xpos1;

dg2 = max_acceptable_dg * max_acceptable_dg;

support = 0;

row1 = y - support_radius;                                          1010
row2 = y + support_radius;

col1 = xpos - support_radius;
col2 = xpos + support_radius;

if (row1 < image.conv[resolution].start)
        row1 = image.conv[resolution].start;
if (row2 > image.conv[resolution].endy)
        row2 = image.conv[resolution].endy;                         1020
if (col1 < image.conv[resolution].start)
        col1 = image.conv[resolution].start;
if (col2 > image.conv[resolution].endx)
        col2 = image.conv[resolution].endx;

for (j = row1; j <= row2; j++)
{
        ydiff2 = j - y;
        ydiff2 = ydiff2 * ydiff2;
        for (i = col1; i <= col2; i++)                              1030
        {
                if (image.map1[i][j].conv[resolution].zc &&
                    !image.map1[i][j].conv[resolution].surface
                    && !(i == (int)xpos && j == y))
                {
                        xpos1 = (i + image.map1[i][j].conv[resolution].pos);
                        xdiff = (xpos - xpos1);
                        xdiff2 = xdiff * xdiff;
                        if (ydiff2 + xdiff2 <= support_radius * support_radius)
                        {                                           1040
                                /*
                                 * loop for all zc's within
                                 * support_radius neighborhood which
                                 * are not matched yet
                                 */
                                sqrt_part = 2 * sqrt(dg2 *
                                                (4 * (xdiff2 + ydiff2) - dg2 * ydiff2));
                                first_part = -4 * disparity + dg2 * (disparity +
                                                2 * xdiff);
                                val1 = xpos1 + (first_part - sqrt_part) /   1050
                                        (-4 + dg2);
                                val2 = xpos1 + (first_part + sqrt_part) /
                                        (-4 + dg2);
                                if (val1 < val2)
```

```
                                        {
                                                search_col1 = (int)val1;
                                                search_col2 = (int)val2;
                                        } else
                                        {
                                                search_col1 = (int)val2;
                                                search_col2 = (int)val1;
                                        }
                                        distance = sqrt(ydiff2 + xdiff2);
                                        if (search_col1 < image.conv[resolution].start)
                                                search_col1 =
                                                        image.conv[resolution].start;
                                        if (search_col2 > image.conv[resolution].endx)
                                                search_col2 =
                                                        image.conv[resolution].endx;
                                        sign1 = image.map1[i][j].conv[resolution].sign;
                                        for (k = search_col1, found = 0;
                                             k <= search_col1 && !found; k++)
                                                /*
                                                 * search the right image for
                                                 * any possible match which
                                                 * would give support to the
                                                 * current candidate match
                                                 */
                                        {
                                                sign2 = image.map2[k][j].
                                                        conv[resolution].sign;
                                                found = image.map2[k][j].
                                                        conv[resolution].zc &&
                                                        !image.map2[k][j].conv[resolution].
                                                        surface && (sign1 == sign2);
                                        }
                                        if (found)
                                                support += 1 / (distance);
                                }
                        }
                }
        }
        return support;
}


/*
 * Pass strips ranges of pixels which fall between consecutive matches in the
 * lowest resolution level.
 */
void    analyze_image(surface_type surface)
{
        row_of_matches_type row;
        matches_type match;
        int     lx1,
                lx2,
                rx1,
                rx2;

        row = surface->rows;
        while (row != NULL)
        {
                match = row->matches;
                rx1 = lx1 = image.conv[3].start - 1;
                while (match != NULL)
                {
```

```
                        lx2 = match->xpos1;
                        rx2 = match->xpos2;
                        analyze_pixels(lx1 + 1, lx2 - 1, rx1 + 1, rx2 - 1, row->row_no);
                        lx1 = lx2;
                        rx1 = rx2;
                        match = match->next;
                }
                lx2 = rx2 = image.conv[3].endx + 1;
                analyze_pixels(lx1 + 1, lx2 - 1, rx1 + 1, rx2 - 1, row->row_no);
                row = row->next;
        }
}

/*
 * Mark pixels in pairs of strips which are wider than MIN_GAP and which have
 * a number of zero-crossings below P_MAX_FRACTION as candidate for intensity
 * matching.
 */
void    analyze_pixels(int lx1, int lx2, int rx1, int rx2, int row_no)
{
        int     sum,
                i;
        float   avg;

        if (lx2 - lx1 + 1 < MIN_GAP || rx2 - rx1 + 1 < MIN_GAP)
                return;
        sum = zc_sum(lx1, lx2, row_no, image.map1);
        avg = (float)sum / (lx2 - lx1 + 1) * P_SIDE;
        if (avg > P_MAX_FRACTION * P_SIDE * P_SIDE)
                return;
        sum = zc_sum(rx1, rx2, row_no, image.map2);
        avg = (float)sum / (lx2 - lx1 + 1) * P_SIDE;
        if (avg > P_MAX_FRACTION * P_SIDE * P_SIDE)
                return;
        for (i = lx1; i <= lx2; i++)
                image.map1[i][row_no].conv[3].zc = 2;
        for (i = rx1; i <= rx2; i++)
                image.map2[i][row_no].conv[3].zc = 2;
}

/*
 * Used by the preceeding function.
 */
int     zc_sum(int x1, int x2, int row_no, map_ptr map)
{
        int     i,
                j;
        int     sum = 0;

        for (i = x1 - P_SIDE / 2; i <= x2 + P_SIDE / 2; i++)
                for (j = row_no - P_SIDE / 2; j <= row_no + P_SIDE / 2; j++)
                {
                        sum += (map[i][j].conv[3].zc == 1) ? 1 : 0;
                }
        return sum;
}

/*
 * Perform intensity matching on a pair of rows. Pass pairs of strips of
 * pixels to function process_segment.
 */
void    process_intensity_row(int y, surface_type surface[5])
```

```
{
        int     col1,                                                   1180
                col2,
                lx1,
                lx2,
                rx1,
                rx2;

        col1 = image.conv[3].start;
        col2 = image.conv[3].endx;

        lx1 = rx1 = col1;                                               1190

        while (lx1 <= col2)
        {
                while (lx1 <= col2 && image.map1[lx1][y].conv[3].zc != 2)
                        lx1++;
                lx2 = lx1;
                while (lx2 <= col2 && image.map1[lx2][y].conv[3].zc == 2)
                        lx2++;
                lx2--;
                                                                        1200
                while (rx1 <= col2 && image.map2[rx1][y].conv[3].zc != 2)
                        rx1++;
                rx2 = rx1;
                while (rx2 <= col2 && image.map2[rx2][y].conv[3].zc == 2)
                        rx2++;
                rx2--;

                if (lx1 < lx2)
                        process_segment(y, lx1, lx2, rx1, rx2, surface);
                                                                        1210
                lx1 = lx2 + 1;
                rx1 = rx2 + 1;
        }
}


/*
 * Perform intensity matching on a pair of pixel strips. A list of candidate
 * matches between pixels in the two strips is constructed and then sorted
 * based on a strength measurement.                                     1220
 */
void    process_segment(int y, int lx1, int lx2, int rx1, int rx2, surface_type surface[5])
{
        zc_list_pixel_type zc_list1,
                zc_list2,
                current_left_zc,
                current_right_zc;
        float   disparity,
                approx_disparity,
                d,                                                      1230
                prev;
        row_of_matches_type row,
                parent_row;
        candidate_matches_pixel_rec candidate_matches_pixel_record;
        candidate_matches_pixel_type candidate_matches;
        candidate_match_pixel_type candidate_match;
        int     i;
        int     resolution = 4;
        int     next;
                                                                        1240
```

```
candidate_matches = &candidate_matches_pixel_record;
candidate_matches->count = 0;
candidate_matches->first = NULL;

zc_list1 = zc_list2 = NULL;

for (i = lx1; i <= lx2; i++)      /* set up list of left strip pixels */
{
        if (!image.map1[i][y].conv[3].surface)
                zc_list1 = add_to_zc_list_pixel(zc_list1, i);                    1250
}
for (i = rx1; i <= rx2; i++)      /* set up list of right strip pixels */
{
        if (!image.map2[i][y].conv[3].surface)
                zc_list2 = add_to_zc_list_pixel(zc_list2, i);
}


row = locate_row(y, resolution, surface);
parent_row = locate_row(y, resolution - 1, surface);                             1260

current_left_zc = zc_list1;
while (current_left_zc != NULL)   /* traverse pixels in the left strip */
{
        approx_disparity = get_approximate_disparity_pixel(current_left_zc, y, parent_row);
        /*
         * compute approximate disparity value based on the disparity
         * values found for lowest resolution level
         */
                                                                                 1270
        current_right_zc = zc_list2;
        while (current_right_zc != NULL)          /* traverse pixels in
                                                   * the right strip */
        {
                /*
                 * compute a correlation difference function between
                 * two squares surrounding the pixels
                 */
                d = difference(current_left_zc->x, current_right_zc->x, y);
                                                                                 1280
                if (d <= MAX_ACCEPTABLE_DIFFERENCE && d != -1.0)
                {
                        {
                                disparity = current_right_zc->x -
                                        current_left_zc->x;
                                if (within_acceptable_range(disparity,
                                                approx_disparity))
                                {
                                        /*
                                         * accept as a candidate                   1290
                                         * match
                                         */
                                        candidate_match =
                                                (candidate_match_pixel_type)
                                                reserve_mem(sizeof(
                                        candidate_match_pixel_rec));
                                        candidate_match->left =
                                                current_left_zc;
                                        candidate_match->right =
                                                current_right_zc;                1300
                                        candidate_match->support =
                                                1 / (d + 1);
```

```
                                        candidate_match->disparity =
                                                disparity;
                                        candidate_match->next =
                                                candidate_matches->first;
                                        candidate_matches->first =
                                                candidate_match;
                                        candidate_matches->count++;
                                }                                                       1310
                        }
                }
                current_right_zc = current_right_zc->next;
        }
        current_left_zc = current_left_zc->next;
}

/* sort candidate matches based on strength */
sort_candidate_matches_pixel(candidate_matches);

                                                                                        1320
prev = 0;
for (i = 0; i < candidate_matches->count; i++)
{

        candidate_match = candidate_matches->sorted[i];
        if (i < candidate_matches->count - 1)
                next = candidate_matches->sorted[i + 1]->support;
        else
                next = 0;
        if (prev != candidate_match->support && next != candidate_match->support)       1330
                /*
                 * if the strength of the candidate match is
                 * different from that of the one before it in the
                 * sorted list and the one after it
                 */
        {
                if (!candidate_match->left->selected &&
                    !candidate_match->right->selected)   /* uniqueness constraint */
                {
                        {                                                               1340
                                add_new_match_pixel(candidate_match, y,
                                        row, surface[resolution], 3);
                                if (row == NULL || row->row_no != y)
                                        row = locate_row(y,
                                                resolution, surface);
                        }
                }
                candidate_match->left->selected = 1;
                candidate_match->right->selected = 1;
        }                                                                               1350
        prev = candidate_match->support;
        delete_mem(candidate_match);
}

if (candidate_matches->count > 0)
        delete_mem(candidate_matches->sorted);

kill_zc_list_pixel(zc_list1);
kill_zc_list_pixel(zc_list2);

}                                                                                       1360

/*
 * Free memory of list of pixels.
```

```
 */
void    kill_zc_list_pixel(zc_list_pixel_type zc_list)
{
        void    *tmp;

        while (zc_list != NULL)
        {
                tmp = zc_list;
                zc_list = zc_list->next;
                delete_mem(tmp);
        }
}

/*
 * Add a new member to the list of pixels
 */
zc_list_pixel_type add_to_zc_list_pixel(zc_list_pixel_type zc_list, int xpos)
{
        zc_list_pixel_type new_zc;

        new_zc = (zc_list_pixel_type) reserve_mem(sizeof(zc_list_pixel_rec));
        new_zc->x = xpos;
        new_zc->selected = 0;
        new_zc->next = zc_list;

        return new_zc;
}

/*
 * Compute the sum of differences between corresponding locations in the two
 * squares surrounding the pair of pixels of the candidate match
 */
float   difference(int x1, int x2, int y)
{
        int     i,
                j,
                diff;
        float   sum = 0;

        if (x1 - D_SIDE / 2 < 0 || x1 + D_SIDE / 2 > image.conv[3].endx + 1 ||
          x2 - D_SIDE / 2 < 0 || x2 + D_SIDE / 2 > image.conv[3].endx + 1 ||
            y - D_SIDE / 2 < 0 || y + D_SIDE / 2 > image.conv[3].endy + 1)
                return -1.0;

        for (i = -D_SIDE / 2; i <= D_SIDE / 2; i++)
                for (j = y - D_SIDE / 2; j <= y + D_SIDE / 2; j++)
                {
                        diff = image.map1[i + x1][j].color - image.map2[i + x2][j].color;
                        sum += abs(diff);
                }
        return sum;
}

/*
 * Sort candidate matches based on strength value.
 */
void    sort_candidate_matches_pixel(candidate_matches_pixel_type candidate_matches)
{
        candidate_match_pixel_type current_match;
        int     i;

        if (candidate_matches->count)
```

```
                candidate_matches->sorted = (candidate_match_pixel_type *) reserve_mem(
                candidate_matches->count * sizeof(candidate_match_pixel_type));
        else
                candidate_matches->sorted = NULL;                                    1430

        current_match = candidate_matches->first;
        i = 0;
        while (current_match != NULL)
        {
                candidate_matches->sorted[i] = current_match;
                i++;
                current_match = current_match->next;
        }

        if (candidate_matches->count)                                                1440
                qsort((void *)candidate_matches->sorted, candidate_matches->count,
                    sizeof(candidate_match_pixel_type), support_comp_pixel);

}

/*
 * Used by the preceeding function.
 */
int     support_comp_pixel(const void *a, const void *b)                            1450
{
        if      ((*(candidate_match_pixel_type *) a)->support >
                        (*(candidate_match_pixel_type *) b)->support)
                return -1;
        if ((*(candidate_match_pixel_type *) a)->support <
            (*(candidate_match_pixel_type *) b)->support)
                return 1;
        return 0;
}

                                                                                     1460

/* Add match to the list of accepted matches */
void    add_new_match_pixel(candidate_match_pixel_type candidate_match, int y,
              row_of_matches_type row, surface_type surface, int resolution)
{
        row_of_matches_type before,
                new_row;
        matches_type current_match,
                match_before,                                                        1470
                new_match;

        if (row == NULL || row->row_no != y)
        {
                before = row;
                if (row != NULL)
                {
                        if (row->row_no < y)
                                row = row->next;
                        else                                                         1480
                                before = row->before;
                }
                new_row = (row_of_matches_type) reserve_mem(sizeof(row_of_matches_rec));
                new_row->row_no = y;
                new_row->matches = NULL;
                new_row->before = before;
                new_row->next = row;
                if (row != NULL)
```

```
                row->before = new_row;
            if (before != NULL)
                before->next = new_row;
            else
                surface->rows = new_row;

            row = new_row;
        }
        current_match = row->matches;
        match_before = NULL;
        while (current_match != NULL && current_match->xpos1 < candidate_match->left->x)
        {
            match_before = current_match;
            current_match = current_match->next;
        }

        if (uncrossed(match_before, candidate_match->left->x,
                candidate_match->right->x) &&
            uncrossed(current_match, candidate_match->left->x,
                candidate_match->right->x))
        {
            new_match = (matches_type) reserve_mem(sizeof(matches_rec));
            new_match->xpos1 = candidate_match->left->x;
            new_match->xpos2 = candidate_match->right->x;
            new_match->disparity = candidate_match->disparity;
            new_match->next = current_match;
            if (match_before)
                match_before->next = new_match;
            else
                row->matches = new_match;


            surface->count++;


            image.map1[candidate_match->left->x][y].conv[resolution].surface = 1;
            image.map2[candidate_match->right->x][y].conv[resolution].surface = 1;
        }
}

/*
 * Compute projected disparity
 */
float   get_approximate_disparity_pixel(zc_list_pixel_type zc, int y, row_of_matches_type row)
{
        float   num,
                dom,
                approx;
        row_of_matches_type current_row;

        num = dom = 0;
        current_row = row;
        while (current_row != NULL && current_row->row_no >= y - disparity_seek_radius)
        {
            row_disparity_approx(current_row, zc->x, y, &num, &dom);
            current_row = current_row->before;
        }

        if (row)
            current_row = row->next;
        else
            current_row = NULL;
```

```
        while (current_row != NULL && current_row->row_no <= y + disparity_seek_radius)
        {
                row_disparity_approx(current_row, zc->x, y, &num, &dom);
                current_row = current_row->next;
        }

        if (dom > 0)
                approx = num / dom;
        else                                                                    1560
                approx = 0;


        return approx;
}

/*
 * Report errors in feature matching
 */
void    error_report(surface_type surface[5])                                   1570
{

        surface_type current_surface;
        row_of_matches_type row;
        matches_type match;
        int     i,
                j,
                diff,
                rr,
                total_zcs,                                                      1580
                correct_disp,
                total_matches,
                incorrect_disparity,
                incorrect_disparity_by_1,
                correct_disparity;

        total_zcs = 0;
        total_matches = 0;
        incorrect_disparity = 0;
        incorrect_disparity_by_1 = 0;                                           1590
        correct_disparity = 0;

        current_surface = surface[3];
        row = current_surface->rows;
        for (j = image.conv[3].start; j <= image.conv[3].endy; j++)
        {
                while (row != NULL && row->row_no < j)
                        row = row->next;
                if (row)
                        match = row->matches;                                   1600
                else
                        match = NULL;

                for (i = image.conv[3].start; i <= image.conv[3].endx; i++)
                {
                        if (image.map1[i][j].conv[3].zc == 1)
                        {
                                total_zcs++;
                                if (image.map1[i][j].conv[3].surface)
                                {                                               1610
                                        total_matches++;
                                        while (match != NULL &&
```

```
                                        (int)(match->xpos1) < i)
                                        match = match->next;
                                if (match == NULL)
                                {
                                        printf("BUG\n");
                                        exit(1);
                                }
                                rr = (int)rint((double)match->xpos1);
                                correct_disp = image.map1[rr][j].correct;
                                if (correct_disp == 99)
                                        incorrect_disparity++;
                                else
                                {
                                        rr = (int)rint((double)match->
                                                        disparity);
                                        diff = abs(correct_disp + rr);
                                        if (diff == 0)
                                                correct_disparity++;
                                        else
                                        if (diff == 1)
                                                incorrect_disparity_by_1++;
                                        else
                                                incorrect_disparity++;
                                }
                        }
                }
        }

        printf("\nFeature matching results:\n");

        printf("Total ZCs: \t\t\t %d\n", total_zcs);
        printf("Total matches: \t\t\t %d \t %4.1f\n", total_matches,
                (total_matches * 100.0 / total_zcs));
        printf("Correct disparity: \t\t %d \t %4.1f \t %4.1f\n", correct_disparity,
                (correct_disparity * 100.0 / total_zcs),
                (correct_disparity * 100.0 / total_matches));
        printf("Wrong disparity by 1: \t\t %d \t %4.1f \t %4.1f\n", incorrect_disparity_by_1,
                (incorrect_disparity_by_1 * 100.0 / total_zcs),
                (incorrect_disparity_by_1 * 100.0 / total_matches));
        printf("Wrong disparity: \t\t %d \t %4.1f \t %4.1f\n", incorrect_disparity,
                (incorrect_disparity * 100.0 / total_zcs),
                (incorrect_disparity * 100.0 / total_matches));
}

/*
 * Report errors in intensity matching
 */
void    error_report_pixel(surface_type surface[5])
{

        surface_type current_surface;
        row_of_matches_type row;
        matches_type match;
        int     diff,
                rr,
                correct_disp,
                total_matches,
                incorrect_disparity,
                incorrect_disparity_by_1,
                correct_disparity;
```

```
        total_matches = 0;
        incorrect_disparity = 0;
        incorrect_disparity_by_1 = 0;
        correct_disparity = 0;

        current_surface = surface[4];                                          1680
        row = current_surface->rows;
        while (row != NULL)
        {
                match = row->matches;
                while (match != NULL)
                {
                        total_matches++;
                        rr = (int)rint((double)match->xpos1);
                        correct_disp = image.map1[rr][row->row_no].correct;
                        if (correct_disp == 99)                                1690
                                incorrect_disparity++;
                        else
                        {
                                rr = (int)rint((double)match->
                                                disparity);
                                diff = abs(correct_disp + rr);
                                if (diff == 0)
                                        correct_disparity++;
                                else
                                if (diff == 1)                                 1700
                                        incorrect_disparity_by_1++;
                                else
                                        incorrect_disparity++;
                        }

                        match = match->next;
                }
                row = row->next;
        }
                                                                               1710

        printf("\nIntensity matching results:\n");

        printf("Total matches: \t\t\t %d\n", total_matches);
        if (total_matches == 0)
                total_matches = 1;
        printf("Correct disparity: \t\t %d \t %4.1f\n", correct_disparity,
                (correct_disparity * 100.0 / total_matches));
        printf("Wrong disparity by 1: \t\t %d \t %4.1f\n", incorrect_disparity_by_1,
                (incorrect_disparity_by_1 * 100.0 / total_matches));          1720
        printf("Wrong disparity: \t\t %d \t %4.1f\n", incorrect_disparity,
                (incorrect_disparity * 100.0 / total_matches));
}
```

Bibliography

[BB81] H. H. Baker and T. O. Binford. Depth from edge and intensity-based stereo. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 631–636, Vancouver, Canada, 1981.

[BD91] C. Brown and C. Dunford. Parallel architectures for fast 3d machine vision. In *3D Model Recognition from Stereoscopic Cues*, pages 67–74, Cambridge, Mass., 1991. MIT Press.

[BJ80] P. Burt and B. Julesz. A disparity gradient limit for binocular fusion. *Science*, 208(9):615–617, May 1980.

[BJ90] P. Burt and B. Julesz. Convolution on mesh connected multicomputers. *IEEE Trans. Pattern Anal. Mach. Intelligence*, PAMI-12(3):315–318, March 1990.

[Bra82] M. Brady. Computational approaches to image understanding. *Computing Surveys*, 14(1):3–71, March 1982.

[CJ90] J. Chang and B. Julesz. Low-level processing of disparity-tuned binocular neurons takes precedence of shape from shading. In *Proceedings of AVRO meeting*, May 1990.

[CM90] J. Chen and G. Medioni. Parallel multiscale stereo matching using adaptive smoothing. In G. Goos and J. Hartmanis, editors, *Lecture Notes in Computer Science: First European Conference on Computer Vision*, pages 99–103, Antibes, France, April 1990. Springer Verlag.

[EW87] R. Eastman and A. Waxman. Using disparity functional for stereo correspondence and surface reconstruction. *Computer Vision, Graphics, and Image Processing*, 39:73–101, 1987.

[Gri81] W. Grimson. *From Images to Surfaces.* MIT Press, Cambridge, Mass., 1981.

[Gri85] W. Grimson. Computational experiments with a feature based stereo algorithm. *IEEE Trans. Pattern Anal. Mach. Intelligence*, PAMI-7(1):17-34, 1985.

[JB91] J. R. Jordan and A. C. Bovik. Using chromatic information in edge-based stereo correspondence. *CVGIP: Image Understanding*, 54(1):98-118, 1991.

[Jul60] B. Julesz. Binocular depth perception of computer-generated patterns. *Bell Systems Tech. J.*, 39:1125-1162, 1960.

[Jul91] B. Julesz. Early vision and focal attention. *Reviews of Modern Physics*, 63(3):735-772, July 1991.

[KB88] N. Kim and A. Bovik. A contour-based stereo matching algorithm using disparity continuity. *Pattern Recognition*, 21(5):505-514, 1988.

[KN77] S. Kuffler and J. Nicholls. *From Neuron to Brain*. Sinauer Press, Sunderland, Mass., 1977.

[MN85] G. Medioni and R. Nevatia. Segment-based stereo matching. *Computer Vision, Graphics, and Image Processing*, 31:2-18, 1985.

[MP79] D. Marr and T. Poggio. A theory of human stereo vision. *Proc. Roy. Soc. London B*, 204:301-328, 1979.

[OK85] Y. Ohta and T. Kanade. Stereo by intra- and inter-scanline search using dynamic programming. *IEEE Trans. Pattern Anal. Mach. Intelligence*, PAMI-7(2):139-154, 1985.

[Ols90] S. I. Olsen. Stereo correspondence by surface reconstruction. *IEEE Trans. Pattern Anal. Mach. Intelligence*, PAMI-12(3):309-315, March 1990.

[PMF85] S. B. Pollard, J. E. Mayhew, and J. P. Frisby. Pmf: a stereo correspondence algorithm using a disparity gradient limit. *Perception*, 14:449-470, 1985.

[Vle93] D. Vleeschauwer. An intensity-based, coarse-to-fine approach to reliably measure binocular disparity. *CVGIP: Image Understanding*, 57(2):204-218, March 1993.

[WB79] H. Wilson and J. Bergen. A four mechanisms model for threshold spatial vision. *Vision Research*, 19:19-32, 1979.

[YGB90] A. L. Yuille, D. Geiger, and H. Bulthoff. Stereo integration, mean field theory and psychophysics. In G. Goos and J. Hartmanis, editors, *Lecture Notes in Computer Science: First European Conference on Computer Vision*, pages 73-82, Antibes, France, April 1990. Springer Verlag.

Vita

- Osama Taleb Abdul-Raheem Masoud
- Born in 1971 Riyadh, Saudi Arabia.
- Received the Bachelor of Science degree in Computer Science in 1992 from King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia.
- Received the Master of Science degree in Computer Science in 1994 from King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia.